IEICE TRANS. INF. & SYST., VOL.E95–D, NO.2 FEBRUARY 2012

PAPER Special Section on Reconfigurable Systems

A Design Method of a Regular Expression Matching Circuit Based on Decomposed Automaton

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

SUMMARY This paper shows a design method for a regular expression matching circuit based on a decomposed automaton. To implement a regular expression matching circuit, first, we convert a regular expression into a non-deterministic finite automaton (NFA). Then, to reduce the number of states, we convert the NFA into a merged-states non-deterministic finite automaton with unbounded string transition (MNFAU) using a greedy algorithm. Next, to realize it by a feasible amount of hardware, we decompose the MNFAU into a deterministic finite automaton (DFA) and an NFA. The DFA part is implemented by an off-chip memory and a simple sequencer, while the NFA part is implemented by a cascade of logic cells. Also, in this paper, we show that the MNFAU based implementation has lower area complexity than the DFA and the NFA based ones. Experiments using regular expressions form SNORT shows that, as for the embedded memory size per a character, the MNFAU is 17.17-148.70 times smaller than DFA methods. Also, as for the number of LCs (Logic Cells) per a character, the MNFAU is 1.56-5.12 times smaller than NFA methods. This paper describes detail of the MEMOCODE2010 HW/SW co-design contest for which we won the first place award.

key words: *regular expression, NFA, DFA, MNFAU, FPGA*

1. Introduction

1.1 Regular Expression Matching for Network Applications

A regular expression represents a set of strings. Regular expression matching detects a pattern represented by a regular expression. Various network applications (e.g., intrusion detection systems [8], [20], a spam filter [21], a virus scanning system [6], and an L7 filter [11]) use regular expression matching. Regular expression matching spends a major part of the total computation time for these applications. The throughput using the perl compatible regular expressions (PCRE) [17] on a general purpose MPU is at most hundreds of Mega bits per second (Mbps) [18], which is too slow. Thus, hardware regular expression matching is required. For network applications, since the high-mix low-volume production and the flexible support for new protocols are required, FPGAs are widely used. Recently, dedicated high-speed transceivers for the high-speed network are embedded in FPGAs. So, we expect extensive use of FPGAs in the future.

Manuscript received April 22, 2011.
Manuscript revised September 2, 2011.

[†]The authors are with the Department of Computer Science and Electronics, Kyushu Institute of Technology, Iizuka-shi, 820–8502 Japan.
a) E-mail: nakahara@aries01.cse.kyutech.ac.jp
b) E-mail: sasao@cse.kyutech.ac.jp
c) E-mail: matsuura@cse.kyutech.ac.jp
DOI: 10.1587/transinf.E95.D.364

Different users require systems with different performance and price. Thus, different architectures should be used. For the IXPs (Internet eXchange Points) and the ISPs (Internet Service Providers), since extremely high throughput (e.g., more than tens of Giga bits per second (Gbps)) is required, such systems tend to have high cost. However, for low-end users, such as SOHO (small office and home office), low cost systems are necessary. The Xilinx FPGA consists of a logic cell (LC) and an embedded memory (BRAM)* [22]. For the Xilinx Spartan III FPGA, the LC consists of a four input look-up table (LUT) and a flip-flop (FF). Since the cost for the FPGA is proportional to the number of LCs, reduction of the number of LCs means reduction of the system cost. In this paper, we propose a design method of the regular expression matching circuit with fewer LCs than conventional methods.

1.2 Proposed Method

The conventional NFA based method uses single-character transitions [19]. In the circuit, each state for the NFA is implemented by an LC. Although a modern FPGA consists of LCs and embedded memories, the conventional NFA based method fails to use available embedded memories (Fig. 1 (a)). In contrast, our previous method uses both LCs and embedded memory to implement the decomposed NFA with string (multi-character) transition [14]. Thus, this method requires fewer LCs than the conventional

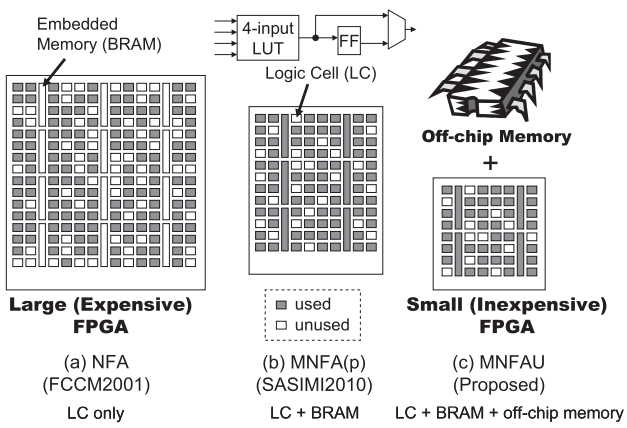

Fig. 1 Platforms for regular expression matching circuits.

*In addition, it consists of a DSP (multiply and accumulation) block, a PLL, a DLL, and a Power PC (embedded processor).

method (Fig. 1 (b)). Moreover, in this paper, to further reduce the FPGA cost, we use the off-chip SRAM to implement most of the regular expression matching circuit. Since the cost for the off-chip SRAM is much lower than for the FPGA, the total cost using an FPGA and the off-chip SRAM is also small. Since our method reduces the FPGA resource drastically, it also reduces the system cost (Fig. 1 (c)).

1.3 Analysis of Complexities of Finite Automata (FAs) on Parallel Hardware Model

Yu et al. [25] compared complexities of the NFA with the DFA on a random access machine (RAM) model. However, to our knowledge, complexities of FAs on the parallel hardware model has not been reported. In this paper, we compare the **non-deterministic finite automaton (NFA)**, the **deterministic finite automaton (DFA)**, and the decomposed NFA with string transition on the parallel hardware model. The decomposed NFA is much smaller than conventional methods.

1.4 Related Work

Regular expressions are detected by finite automata. In a DFA, for each state and each input, there is a unique transition, while in a NFA, for each state for each input, multiple transitions may exist. In an NFA, there exist **ε-transitions** to other states without consuming input characters. Various DFA-based regular expression matchings exist: An Aho-Corasick algorithm [1]; a bit-partition of the Aho-Corasick DFA by Tan et al. [23]; a combination of the bit-partitioned DFA and the MPU [3]; and a pipelined DFA [5]. Also, various NFA-based regular expression matchings exist: an algorithm that emulates the NFA (Baeza-Yates's NFA) by shift and AND operations on a computer [2]; an FPGA realization of Baeza-Yates's NFA (Sidhu-Prasanna method) [19]; prefix sharing of regular expressions [12]; and a method that maps repeated parts of regular expressions to the Xilinx FPGA primitive (SRL16) [4].

1.5 Organization of the Paper

The rest of the paper is organized as follows: Section 2 shows a regular expression matching circuit based on the finite automaton; Section 3 shows a regular expression matching circuit based on an NFA with string transition; Section 4 shows a design method of a regular expression matching circuit based on an NFA with string transition; Section 5 compares complexities on the parallel hardware model; Section 6 shows the experimental results; Section 7 shows the result of MEMOCODE 2010 HW/SW co-design contest; and Sect. 8 concludes the paper.

This paper is an extension of previous publications [13]–[15].

2. Regular Expression Matching Circuit Based on Automaton

2.1 Regular Expression

A regular expression consists of characters and meta characters. A character is represented by eight bits. The length of the regular expression is the number of characters. Table 1 shows meta characters considered in this paper. Note that, in Table 1, r denotes a regular expression.

2.2 Regular Expression Matching Circuit Based on Deterministic Finite Automaton

Definition 2.1: A deterministic finite automaton (DFA) consists of a five-tuple $M_{DFA} = (S, \Sigma, \delta, s_0, A)$, where $S = \{s_0, s_1, \ldots, s_{q-1}\}$ is a finite set of states; Σ is a finite set of input characters; δ is a transition function ($\delta : S \times \Sigma \to S$); $s_0 \in S$ is the initial state; and $A \subseteq S$ is the set of accept states. Since our system accommodates ASCII characters, it is convenient to choose $|\Sigma| = 2^8 = 256$.

Definition 2.2: Let $s \in S$, and $c \in \Sigma$. If $\delta(s, c) \in S$, then c denotes a transition character from state s to state $\delta(s, c)$.

To define a transition string accepted by the DFA, we extend the transition function δ to $\hat{\delta}$.

Definition 2.3: Let Σ^+ be a set of strings, and $\hat{\delta} : S \times \Sigma^+ \to S$ be the extended transition function. If $C \subseteq \Sigma^+$ and $s \in S$, then $\hat{\delta}(s, C)$ represents a transition state of s with respect to the input string C.

Definition 2.4: Suppose that $M_{DFA} = (S, \Sigma, \delta, s_0, A)$. Let $C_{in} \subseteq \Sigma^+$. Then, M_{DFA} accepts a string C_{in}, if the following relation holds:

$$\hat{\delta}(s_0, C_{in}) \in A. \tag{1}$$

Let c_i be a character of a string $C = c_0 c_1 \cdots c_n$, and δ be a transition function. Then, the extended transition function $\hat{\delta}$ is defined recursively as follows:

$$\hat{\delta}(s, C) = \hat{\delta}(\delta(s, c_0), c_1 c_2 \cdots c_n). \tag{2}$$

From (1) and (2), the DFA performs the string matching by repeating state transitions.

Table 1 Meta characters for perl compatible regular expression considered in this paper.

Regular Expression	Meaning
$r_1 r_2$	concatenation (r_1 followed by r_2)
$r_1 \mid r_2$	r_1 or r_2 (union)
$r*$	repeat r zero or more times (Kleene closure)
$r+$	repeat r one or more times
$r?$	repeat r zero or one time
r{n,m}	repeat r at least n and at most m times
.	match any single character except newline (\n)
[]	set of characters
[^]	complement set of characters
^	matching start from the first character
$	matching ends at the last character

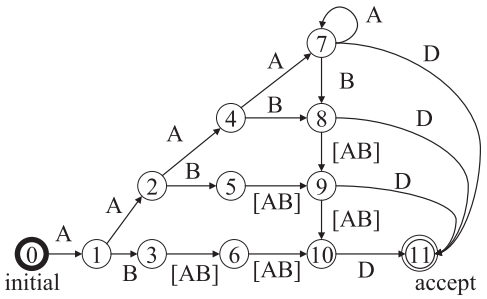

Fig. 2 DFA for the regular expression "A+[AB]{3}D".

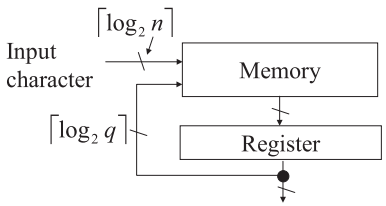

Fig. 3 DFA machine.

Example 2.1: Figure 2 shows the DFA for the regular expression "A+[AB]{3}D". Note that, "A+" denotes one or more "A", and "[AB]{3}=[AB][AB][AB]" denotes three times occurrences of "A" or "B". ■

Example 2.2: Consider the string matching for an input "AABAD" using the DFA shown in Fig. 2. Note that, [AB] denotes "A" or "B", while "AB" denotes the concatenation "A" and "B". [AB][CD] denotes the set of four strings {AC,AD,BC,BD}. Let s_0 be the initial state. First, $\delta(s_0, A) = s_1$. Second, $\delta(s_1, A) = s_2$. Third, $\delta(s_2, B) = s_5$. Fourth, $\delta(s_5, A) = s_9$. Finally, $\delta(s_9, D) = s_{11}$. Since the state s_{11} is an accept state, the string "AABAD" is accepted. ■

Figure 3 shows the DFA machine, where the register stores the present state and the memory realizes the transition function δ. Let $q = |S|$ be the number of states, and $n = |\Sigma|$ be the number of characters in Σ. Then, the amount of memory to implement the DFA is $\lceil \log_2 q \rceil 2^{\lceil \log_2 n \rceil + \lceil \log_2 q \rceil}$ bits†.

2.3 Regular Expression Matching Circuit Based on Nondeterministic Finite Automaton

Definition 2.5: A non-deterministic finite automaton (NFA) consists of a five-tuple $M_{NFA} = (S, \Sigma, \gamma, s_0, A)$, where S, Σ, s_0, and A are the same as Definition 2.1, while the transition function $\gamma : S \times (\Sigma \cup \{\varepsilon\}) \to P(S)$ is different. Note that, ε denotes an empty character, and $P(S)$ denotes the power set of S.

In the NFA, the empty (ε) input is permitted. Thus, a state for the NFA can transit to multiple states. The state transition with the ε input denotes an ε transition. In this paper, in a state transition diagram, an ε symbol with an arrow denotes the ε transition.

Example 2.3: Figure 4 shows the NFA for the regular expression for "A+[AB]{3}D", and also shows the states visited when the input string is "AABAD". Note that, multiple state transitions occur in certain rows, since the NFA can be in multiple states given input string "AABAD". There is at least one path from the initial state s_0 to the accept state s_5. Thus, "AABAD" is accepted by this NFA. ■

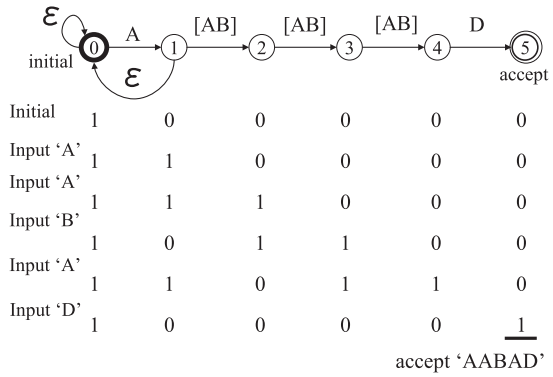

Fig. 4 NFA for the regular expression "A+[AB]{3}D".

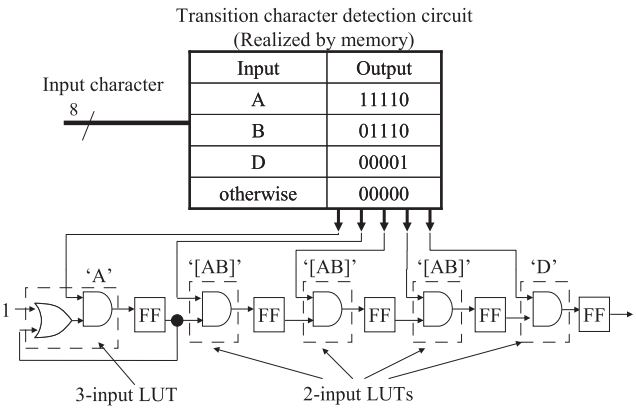

Fig. 5 A circuit for the NFA shown in Fig. 4.

Sidhu and Prasanna [19] realized an NFA with single-character transitions for regular expressions [2]. Figure 5 shows the circuit for the NFA. To realize the NFA, first, the memory detects the character for the state transition, and then character detection signals are sent to small machines that correspond to states of the NFA. Each small machine is realized by a flip-flop and an AND gate. Also, an ε-transition is realized by OR gates and interconnections on the FPGA. Then, machines for the accepted states generate the match signal.

3. Regular Expression Matching Circuit Based on NFA with String Transition

3.1 MNFAU

Sidhu-Prasanna's method [19] does not use embedded mem-

†Since the size of the register in the DFA machine is much smaller than that for the memory storing the transition function, we ignore the size of the register.

ory[†]. So, their method is inefficient with respect to the resource utilization of an FPGA, since a modern FPGA consists of LCs and embedded memories. In the circuit for the NFA, each state is implemented by an LC of an FPGA. Thus, the necessary number of LCs increases with the number of states. To reduce the number of states, we propose a regular expression matching circuit based on a merged-states non-deterministic finite automaton with unbounded string transition (MNFAU). To convert an NFA into an MNFAU, we merge a sequence of states. However, to retain the equivalence between the NFA and the MNFAU, we merge the states as follows:

Lemma 3.1: Let $S = \{s_0, s_1, \ldots, s_{q-1}\}$ be the set of states for the NFA. Assume that for a subset $S' = \{s_k, s_{k+1}, \ldots, s_{k+p-1}\} \subseteq S$, where $k \leq i \leq k+p-2$, s_i goes to s_{i+1} only. Then, the states in S' are merged into one state of the MNFAU only if both the in-degree and the out-degree for s_i $(k \leq i \leq k+p-1)$ are one.

Definition 3.1: Suppose that a set of states $\{s_k, s_{k+1}, \ldots, s_{k+p}\}$ of an NFA is merged into a state S_M of an MNFAU. A string $C = c_k c_{k+1} \cdots c_{k+p}$ is **a transition string of** S_M, where $c_j \in \Sigma$ is a transition character of s_j for $j = k, k+1, \ldots, k+p$.

Example 3.1: In the NFA shown in Fig. 4, the set of states $\{s_2, s_3, s_4, s_5\}$ can be merged into a state of the MNFAU. However, the set of states $\{s_1, s_2\}$ cannot be merged, since $e_1 \neq 0$. ■

Example 3.2: Figure 6 shows possible MNFAUs derived from the NFA shown in Fig. 4. In the NFA, since three states $\{s_2, s_3, s_4\}$ have $e_i = 0$, the number of possible MNFAUs are eight. In Fig. 6, the MNFAU (h) is the most compact MNFAU. ■

As shown in Example 3.2, the conversion of the compact MNFAU from the given NFA exist. However, we must consider the restriction for the hardware. The next Section shows the hardware realization for the MNFAU, and Sect. 4 shows the design method for the MNFAU.

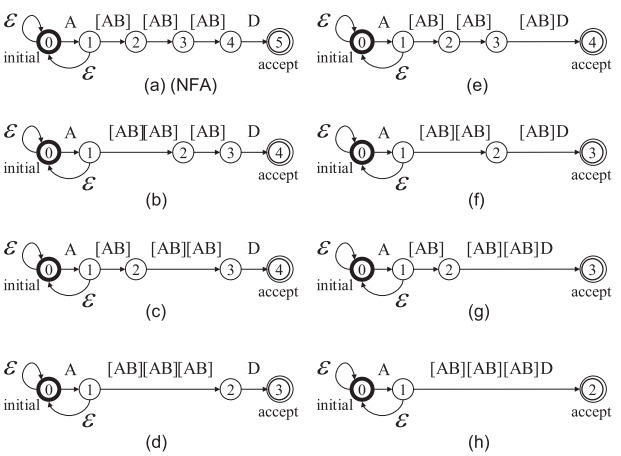

Fig. 6 Possible MNFAUs derived from the NFA shown in Fig. 4.

3.2 Realization of MNFAU

An MNFAU is decomposed into a DFA and an NFA. The DFA is realized by the transition string detection circuit, and the NFA is realized by the state transition circuit. Figure 7 shows a decomposed MNFAU. Since transition strings do not include meta characters[††], they are detected by exact matching. Exact matching is a subclass of regular expression matching and the DFA can be realized by a feasible amount of hardware [25]. On the other hand, the state transition part treating the ε transition is implemented by the cascade of logic cells shown in Fig. 5.

3.2.1 Transition String Detection Circuit

Since each state of the MNFAU consists of different number of states of the NFA, lengths of the transition strings for states of the MNFAU are different. To detect multiple strings with different lengths, we use the Aho-Corasick DFA (AC-DFA) [1]. To obtain the AC-DFA, first, the transition strings are represented by a text tree (Trie). Next, the failure paths that indicate the transitions for the mismatches are attached to the text tree. Since the AC-DFA stores failure paths, no backtracking is required. By scanning the input only once, the AC-DFA can detect all the strings represented by the regular expressions. The AC-DFA is realized by the circuit shown in Fig. 3. Let $q = |S|$ be the number of states, and $n = |\Sigma|$ be the number of characters in Σ. Then, the amount of memory to implement the AC-DFA is $\lceil \log_2 q \rceil 2^{\lceil \log_2 n \rceil + \lceil \log_2 q \rceil}$ bits.

Example 3.3: Figure 8 illustrates the AC-DFA accepting transition strings "A" and "[AB][AB][AB]D" for the MNFAU shown in Fig. 6. ■

3.2.2 State Transition Circuit [15]

In an NFA, each state is realized by the small machine consisting of a flip-flop and an AND gate. Figure 9 shows the state transition circuit for the MNFAU. When the AC-DFA detects the transition string ("ABD" in Fig. 9), a detection signal is sent to the state transition circuit.Then, the state

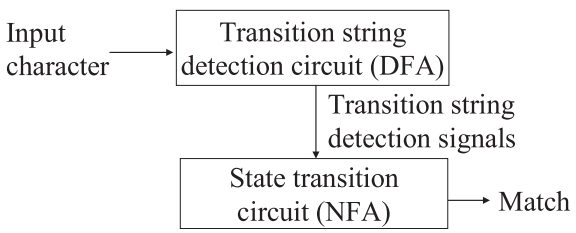

Fig. 7 Decomposed MNFAU.

[†]Their method uses single character detectors (comparators) instead of the memory shown in Fig. 5.

[††]However, a meta character "[]" can be used.

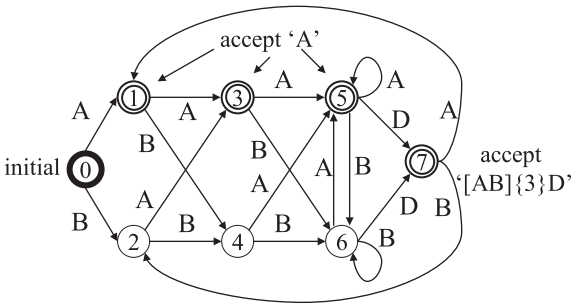

Fig. 8 AC-DFA accepting strings "A" and "[AB][AB][AB]D".

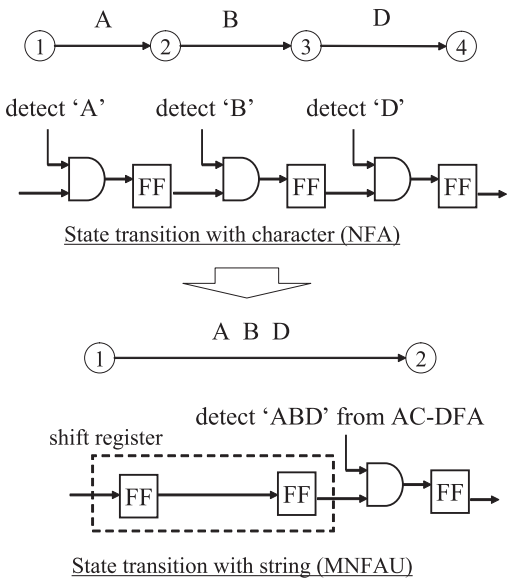

Fig. 9 State transition circuit for the MNFAU.

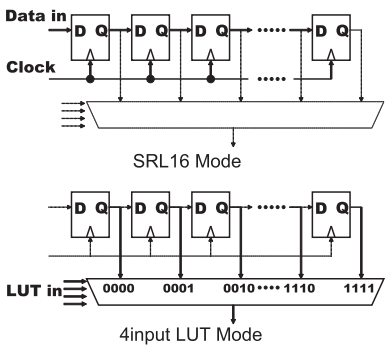

Fig. 10 Two LUT modes for Xilinx FPGA.

transition is performed. The AC-DFA scans a character in every clock, while the state transition requires p clocks to perform the state transition, where p denotes the length of the transition string. Thus, a $(p-1)$-bit shift register is inserted between small machines to synchronize with the AC-DFA (In Fig. 9, a two-bit shift register is inserted). A four input LUT of a Xilinx FPGA can also be used as a shift register with up to 16 bits (SRL16) [24]. Figure 10 shows two LUT modes of a Xilinx FPGA[†]. With the SRL16, we can

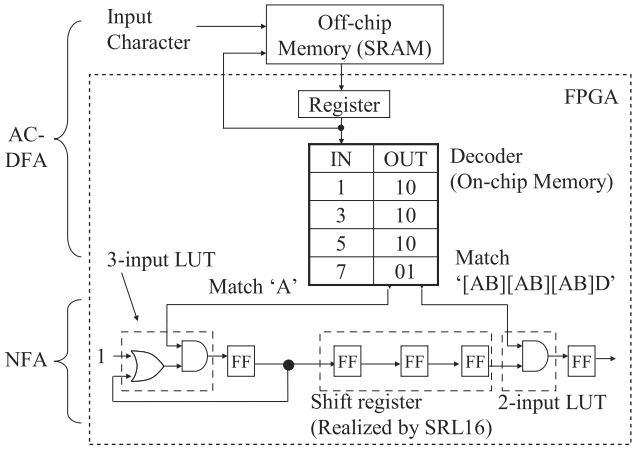

Fig. 11 An example circuit for the decomposed MNFAU.

reduce the necessary number of LUTs and flip-flops.

Figure 11 shows an example circuit for the decomposed MNFAU. We decompose the MNFAU into the transition string detection circuit and the state transition circuit. The transition function for the AC-DFA is realized by the off-chip memory (i.e., SRAM), while other parts are realized by the FPGA. In the AC-DFA, a register with $\lceil \log_2 q \rceil$ bits shows the present state, where q is the number of states for the AC-DFA. On the other hand, a u-bit detection signal is necessary for the state transition circuit, where u is the number of states for the MNFAU. We use a decoder that converts a $\lceil \log_2 q \rceil$-bit state to a u-bit detection signal. Since the decoder is relatively small, it is implemented by the embedded memory[††] in the FPGA.

Example 3.4: In Fig. 11, the address for the decoder memory corresponds to the assigned state number for the AC-DFA shown in Fig. 8. The decoder memory produces the detection signal for the state transition circuit. As for the NFA based regular expression matching circuit shown in Fig. 5, the number of LUTs is five, and the number of FFs is five. On the other hand, as for the MNFAU based regular expression matching circuit shown in Fig. 11, the number of LUTs is three, and the number of FFs is two. ■

4. Design of a Decomposed Regular Expression Matching Circuit

4.1 Design Flow

This section shows the design method for the decomposed regular expression matching circuit. Figure 12 shows the design flow. First, the NFA is constructed from the given

[†]In the Xilinx Spartan III FPGA, a CLB consists of four SLICEs, and a SLICE consists of two LCs. In the CLB, two SLICEs can be configured as an SRL16 or an LUT, while other two SLICEs can be configured as an LUT only.

[††]It can also be implemented by LUTs. However, when q is large, it requires a large number of LUTs. In our experiment for the SNORT, $q = 10,066$.

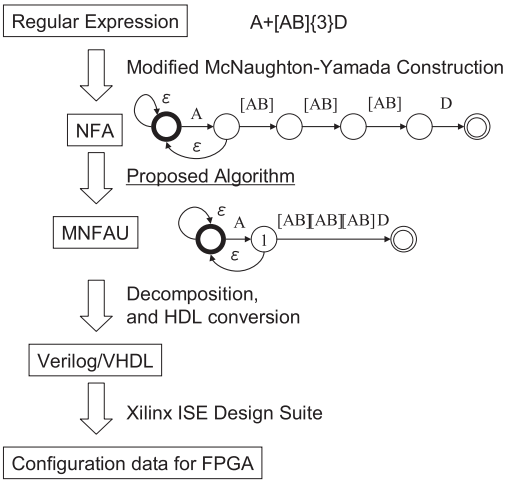

Fig. 12 Design flow for regular expression matching circuit.

regular expression. Then, it is converted into an MNFAU. The conversion method is described in Sect. 4.3. Next, the MNFAU is decomposed into the DFA part and the NFA part. The DFA part is realized by the sequencer shown in Fig. 3 and a decoder, while the NFA part is realized by the cascade of LCs shown in Fig. 9. Then, the decomposed MNFAU is converted into the HDL source file. Finally, we use the Xilinx ISE Design Suite, an FPGA synthesis tool to generate the configuration data for the FPGA.

4.2 Construction of the NFA

The regular expression shown in Table 1 satisfies the following relations:

$$\begin{aligned} r+ &= rr* \\ r? &= (\varepsilon \mid r) \\ r\{n,m\} &= \overbrace{r\cdots r}^{n} \mid \overbrace{r\cdots r}^{n+1} \mid \cdots \mid \overbrace{r\cdots r}^{m} \\ [c_i c_j] &= c_i \mid c_j, \end{aligned}$$

where r denotes a regular expression, $c_i \in \Sigma$ denotes a character, Σ denotes the set of characters, ε denotes an empty character, and c_i, c_j denotes distinct characters. Thus, to construct an NFA from a regular expression, it is sufficient to consider the "state transition with a character", "concatenation", "Kleene closure (*)", and "union (|)". To construct the NFA from the given regular expression, we use the modified McNaughton-Yamada construction [9]. Figure 13 shows the modified McNaughton-Yamada construction.

4.3 Design Algorithm for the Decomposed MNFAU

As shown in Example 3.2, any NFA can be converted into an MNFAU satisfying Algorithm 3.1. Thus, the conversion into a compact MNFAU is important. The conversion problem to an MNFAU from an NFA is formulated as follows:

Problem 4.1: Let $S = \{s_0, s_1, \ldots, s_{q-1}\}$ be the set of states

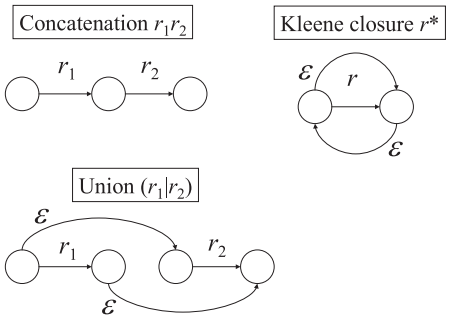

Fig. 13 Modified McNaughton-Yamada constructions.

of the NFA; t be the number of states for the NFA with $e_i > 0$ $(0 \le i \le q-1)$, where e_i be the total number of ε transition inputs and outputs in the state s_i; $(S_1, S_2, \ldots, S_u)$ be a partition of S, where $S_i \subseteq S$ and $S_i \cap S_j = \phi (i \neq j)$; C_i be a transition string for a set of states S_i; $C = \{C_1, C_2, \ldots, C_u\}$ be a set of transition strings; $M(C)$ be the memory size of the AC-DFA for C; and $M_{off-chip}$ be the memory size for the off-chip memory. Then, find a partition S that minimizes u satisfying the memory constraint $M(C) < M_{off-chip}$, where u is the number of partitions in S. Note that, for each $S_i = \{s_k, s_{k+1}, \ldots, s_{k+p}\}$, $e_i = 0$ for $i = k, k+1, \cdots, k+p-1$.

Since the number of possible MNFAUs 2^{q-t-1} can be very large, an exhaustive method to find a minimum MNFAU satisfying the off-chip memory constraint $M(C) < M_{off-chip}$ is impractical. In this paper, we propose a greedy method to find a near minimum MNFAU.

Algorithm 4.1: (Find a near minimum MNFAU from the NFA)
Let $S = \{s_0, s_1, \ldots, s_{q-1}\}$ be a set of states for the NFA, and $M_{off-chip}$ be the memory size for the off-chip memory.

1. Obtain a minimum partition $S = S_1 \cup S_2 \cup \cdots \cup S_u$, where $S_i \cap S_j = \phi (i \neq j)$, such that, for each $S_i = \{s_k, s_{k+1}, \ldots, s_{k+p}\}$, $e_i = 0$ for $i = k, k+1, \cdots, k+p-1$. Then, obtain a set of transition strings $C = \{C_1, C_2, \ldots, C_u\}$.
2. Construct the AC-DFA for C. Then, obtain $M(C)$.
3. If $M(C) \le M_{off-chip}$, then go to Step 6.
4. Select the maximum $S_i = \{s_k, s_{k+1}, \ldots s_{k+n}\}$ from S, and partition it into two subsets S_{i_1} and S_{i_2}, where $S_{i\_1} = \{s_k, s_{k+1}, \ldots, s_{k+\lceil \frac{n}{2} \rceil}\}$ and $S_{i\_2} = \{s_{k+\lceil \frac{n}{2} \rceil+1} \ldots, s_{k+n}\}$. Also, obtain a set of transition strings $C = \{C_1, C_2, \ldots, C_{i\_1}, C_{i\_2}, \ldots, C_u\}$, where C_{i_1} is a transition string for S_{i_1}, and C_{i_2} is that for S_{i_2}.
5. Go to Step 2.
6. Terminate the algorithm.

Algorithm 4.1 partitions the maximum subset of S until the memory size $M(C)$ does not exceed $M_{off-chip}$.

5. Complexity of Regular Expression Matching Circuit on Parallel Hardware Model

The Xilinx FPGA consists of **logic cells (LCs)** and embed-

ded memories. An LC consists of a four input look-up table (LUT) and a flip-flop (FF) [22]. Therefore, as for the area complexity, we consider both the LC complexity and the embedded memory complexity.

5.1 Theoretical Analysis

5.1.1 Aho-Corasick DFA

As shown in Fig. 3, a machine for the DFA consists of a register storing the present state, and the memory for the state transition. The DFA machine reads one character and computes the next state in every clock. Thus, the time complexity is $O(1)$. Also, since the size of the register is fixed, the LC complexity is $O(1)$. Yu et al. [25] showed that, for m regular expressions with length s, the memory complexity is $O(|\Sigma|^{sm})$ for the Aho-Corasick DFA, where $|\Sigma|$ denotes the number of characters in Σ.

5.1.2 Baeza-Yates NFA

As shown in Fig. 5, an NFA consists of the memory for the transition character detection, and a cascade of LCs each of which consists of an LUT (realizing AND and OR gates) and a FF. Thus, for m regular expressions with length s, the LC complexity is $O(ms)$. Since the amount of memory for the transition character detection is $m \times |\Sigma| \times s$, the memory complexity is $O(ms)$. A regular expression matching circuit based on an NFA has s states and processes one character every clock, including ε transitions. By using m circuits shown in Fig. 5, the circuit can match m regular expressions in parallel. Thus, the time complexity is $O(1)$.

5.1.3 Decomposed MNFAU

As shown in Fig. 7, the decomposed MNFAU consists of a transition string detection circuit and a state transition circuit. The transition string detection circuit is realized by the DFA machine shown in Fig. 3. Let p_{max} be the maximum length of the transition string in the MNFAU, and $|\Sigma|$ be the number of characters in the set Σ. From the analysis of the DFA [7], the memory complexity is $O(|\Sigma|^{p_{max}})$, while the LC complexity for the AC-DFA machine is $O(1)$. The state transition circuit is realized by the cascade of LCs shown in Fig. 11. Let p_{ave} be the average number of merged states in the NFA, s be the length of the regular expression, and m be the number of regular expressions. Since one state in the MNFAU corresponds to p_{ave} states in the NFA, the LC complexity is $O(\frac{ms}{p_{ave}})$. By using m parallel circuits, the circuit matches m regular expressions in parallel. Thus, the time complexity is $O(1)$.

Note that, in most cases, the NFA requires longer word length than the MNFAU. The NFA requires sm-bit words, while the MNFAU requires $\lceil \log_2 q \rceil$-bit words[†], where q is the number of states for the MNFAU. For the NFA, off-chip memories are hard to use, since the FPGA has a limited number of pins. Thus, the NFA requires a large number of on-chip memories. On the other hand, for the MNFAU, off-chip memory is easy to use, since the required number of pins is small. Although the MNFAU requires larger memory than the NFA, the MNFAU can use off-chip memory and a small FPGA. This reduces the hardware cost.

Table 2 Complexities for the NFA, the DFA, and the decomposed MNFAU on the parallel hardware mode.

	Time	Area	
		Memory	#LC
Baeza-Yates's NFA	$O(1)$	$O(ms)$	$O(ms)$
Aho-Corasick DFA	$O(1)$	$O(\|\Sigma\|^{ms})$	$O(1)$
Decomposed MNFAU	$O(1)$	$O(\|\Sigma\|^{p_{max}})$	$O(\frac{ms}{p_{ave}})$

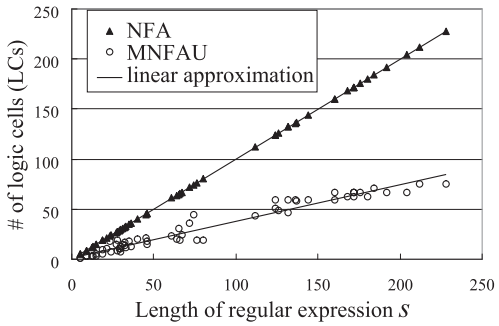

Fig. 14 Relation between the length s of regular expression and the number of LCs.

Table 2 compares the area and time complexities for the NFA, the DFA, and the decomposed MNFAU on the parallel hardware model. As shown in Table 2, by using the decomposed MNFAU, the memory size is reduced to $\frac{1}{|\Sigma|^{ms-p_{max}}}$ of the DFA, and the number of LCs is reduced to $\frac{1}{p_{ave}}$ of the NFA.

5.2 Analysis Using SNORT

To verify the analysis of the previous part, we compared the memory size and the number of LCs for practical regular expressions. We selected 80 regular expressions from the intrusion detection system SNORT [20], and for each regular expression, we generated the DFA, the NFA, and the decomposed MNFAU. Then, we obtained the number of LCs and the memory size. Figure 14 shows the relation between the length of the regular expression s and the number of LCs, while Fig. 15 shows the relation between s and the memory size. Note that, Fig. 14, has a linear vertical axis, while Fig. 15 has a logarithmic vertical axis. As shown in Fig. 14, the ratio between the number of LCs and s is a constant. On the other hand, as shown in Fig. 15, the ratio between the memory size and s increases exponentially.

Therefore, both the theoretical analysis and the experiment using SNORT show that the decomposed MNFAU realizes regular expressions efficiently.

[†]For example, in the SNORT, the value of sm is about 100,000, while $\lceil \log_2 q \rceil = 14$.

Table 3 Comparison with other methods.

Method	FA Type	FPGA	Th (Gbps)	#LC	MEM (Kbits)	#Char	#LC/ #Char	MEM/ #Char
Pipelined DFA [5] (ISCA'06)	DFA	Virtex 2	4.0	247,000	3,456	11,126	22.22	3182.2
MPU+Bit-partitioned DFA [3] (FPL'06)	DFA	Virtex 4	1.4	N/A	6,000	16,715	N/A	367.5
Improvement of Sidhu-Prasanna method [4] (FPT'06)	NFA	Virtex 4	2.9	25,074	0	19,580	1.28	0
MNFA(3) [14] (SASIMI'10)	MNFA(p)	Virtex 6	3.2	4,707	441	12,095	0.39	37.3
MNFAU (Proposed method)	MNFAU	Spartan 3	1.6	19,552	1,585	75,633	0.25	21.4

Table 4 Result of MEMOCODE2010 HW/SW co-design contest.

Team Name	Place	Number of Regular Expressions	Performance (Mbps)	Platform	Institution
Sasao Lab (Our team)	1 (tie)	140	798	FPGA: Altera Stratix III	Kyushu Institute of Technology, Japan
Limenators	1 (tie)	140	500	FPGA: Xilinx V5LX330	IBM Research, USA
SpbSU	3	126	500	FPGA: Xilinx ML505	Lanit-Tercom, Russia
Kraaken	4	85	734	FPGA: Xilinx XUPV5	AMD, USA
Battery	5	35	524	GPU: NVIDIA Tesla T10	Iowa State University, USA
Team IISC	6	25	584	FPGA: Xilinx XUPV5	Indian Institute of Science, India
Tosan	7	25	524	GPU: NVIDIA GTX 295	Sharif University of Technology, Iran
[Ii][Ss][Uu][0-2]{4}	8	42	534	FPGA: Altera Stratix III	Iowa State University, USA

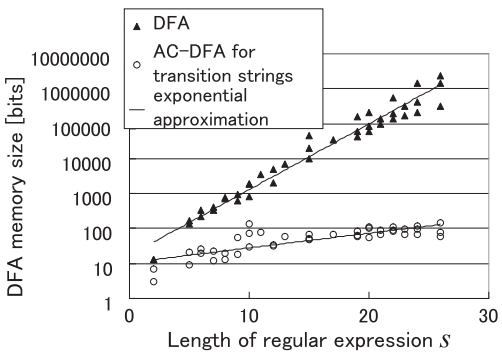

Fig. 15 Relation between the length s of regular expression and the memory size.

6. Experimental Results

6.1 Implementation of the MNFAU

We selected regular expressions from SNORT (open-source intrusion detection system), and generated the decomposed MNFAU. Then, we implemented these on the Xilinx Spartan III FPGA (XC3S4000: 62,208 logic cells (LCs), total 1,728 Kbits BRAM). The total number of regular expressions is 1,114 (75,633 characters). The number of states for the MNFAU is 12,673, and the number of states for the AC-DFA for the transition string is 10,066. This implementation requires 19,552 LCs, and an off-chip memory of 16 Mbits. Note that, the 16 Mbits off-chip SRAM is used to store the transition function of the AC-DFA, while 1,585 Kbits on-chip BRAM is used to realize the decoder. The FPGA operates at 271.2 MHz. However due to the limitation on the clock frequency by the off-chip SRAM, the system clock was set to 200 MHz. Our regular expression matching circuit scans one character in every clock. Thus, the throughput is $0.2 \times 8 = 1.6$ Gbps.

6.2 Comparison with Other Methods

Table 3 compares our method with other methods. In Table 3, *Th* denotes the throughput (Gbps); *#LC* denotes the number of logic cells; *MEM* denotes the amount of embedded memory for the FPGA (Kbits); and *#Char* denotes the number of characters for the regular expression. Table 3 shows that, as for the embedded memory size per a character, the MNFAU requires 17.17-148.70 times smaller memory than the DFA method. Also, as for the number of LCs per a character, the MNFAU requires 1.56-5.12 times fewer LCs than the NFA method.

7. Result of Eighth MEMOCODE2010 HW/SW Co-design Contest [16]

In July 2010, the eighth ACM/IEEE international conference on formal methods and models for co-design (MEMOCODE2010) challenged teams to implement the architecture for an unique type of a deep packet inspector called CANSCID (Combined Architecture for Stream Categorization and Intrusion Detection). Metrics judging the design are:

1. The number of category patterns and the intrusion patterns represented by regular expressions.
2. The system throughput must be higher than the line rate of 500 Mbps.

We implemented CANSCID on a Terasic Technologies Inc. DE3 development board utilizing an Altera Stratix III FPGA (EP3S340H1152C3N4). We realized 140 regular expression patterns of the design contest using MNFA (3)† [15]. Table 4 shows the result of design con-

†MNFA (3) is a special case of MNFAU whose transition string has at most three characters [14]. After the design contest, we generalized MNFA (3) to the MNFAU. Although the MNFA (3) is easy to generate, it requires more hardware than the MNFAU.

test. Team Sasao Lab (our team) and Limenators were joint winners, each implementing 140 patterns while maintaining a line rate of 500 Mbps. Only our team used the MNFA (3) approach rather than DFAs for the regular expression matching. In this way, we could implement regular expressions compactly while maintaining the highest speed.

8. Conclusion

In this paper, we proposed a regular expression matching circuit based on a decomposed MNFAU. To implement the circuit, first, we converted the regular expressions into an NFA. Then, to reduce the number of states, we converted the NFA into an MNFAU by a greedy method. Next, to realize it by a feasible amount of the hardware, we decomposed the MNFAU into a transition string detection part and a state transition part. The transition string detection part was implemented by an off-chip memory and a simple sequencer, while the state transition part was implemented by a cascade of logic cells. Also, this paper showed that the MNFAU based implementation has lower area complexity than the DFA and the NFA based ones. The implementation of SNORT showed that, as for the embedded memory size per a character, the MNFAU is 17.17-148.70 times smaller than DFA methods. Also, as for the number of LCs per a character, the MNFAU is 1.56-5.12 times smaller than NFA methods. With MNFA (3), we won the first place award in the MEMOCODE2010 HW/SW co-design contest.

Acknowledgments

This research is supported in part by the grant of Regional Innovation Cluster Program (Global Type, 2nd Stage). Discussion with Prof. J. T. Butler was quite useful. Dr. Hiroaki Yoshida encouraged us to participate the design contest. The hard work of the organizers of the MEMOCODE2010 HW/SW co-design contest is also appreciated.

References

[1] A.V. Aho and M.J. Corasick, "Efficient string matching: An aid to bibliographic search," Commun. ACM, vol.18, no.6, pp.333–340, 1975.
[2] R. Baeza-Yates and G.H. Gonnet, "A new approach to text searching," Commun. ACM, vol.35, no.10, pp.74–82, Oct. 1992.
[3] Z.K. Baker, H. Jung, and V.K. Prasanna, "Regular expression software deceleration for intrusion detection systems," FPL'06, pp.28–30, 2006.
[4] J. Bispo, I. Sourdis, J.M.P. Cardoso, and S. Vassiliadis, "Regular expression matching for reconfigurable packet inspection," FPT'06, pp.119–126, 2006.
[5] B.C. Brodie, D.E. Taylor, and R.K. Cytron, "A scalable architecture for high-throughput regular-expression pattern matching," ISCA'06, pp.191–202, 2006.
[6] "Clam anti virus: open source anti-virus toolkit," http://www.clamav.net/lang/en/
[7] R. Dixon, O. Egecioglu, and T. Sherwood, "Automata-theoretic analysis of bit-split languages for packet scanning," CIAA'08, pp.141–150, 2008.
[8] "Firekeeper: Detect and block malicious sites," http://firekeeper.mozdev.org/
[9] T. Ganegedara, Y.E. Yang, and V.K. Prasanna, "Automation framework for large-scale regular expression matching on FPGA," FPL2010, pp.50–55, 2010.
[10] Z. Kohavi, Switching and Finite Automata Theory, McGraw-Hill, 1979.
[11] "Application layer packet classifier for linux," http://l7-filter.sourceforge.net/
[12] C. Lin, C. Huang, C. Jiang, and S. Chang, "Optimization of regular expression pattern matching circuits on FPGA," DATE'06, pp.12–17, 2006.
[13] H. Nakahara, T. Sasao, and M. Matsuura, "A regular expression matching circuit based on a decomposed automaton," ARC'11, Lecture Notes in Computer Science, no.6578, pp.16–28, March 2011.
[14] H. Nakahara, T. Sasao, and M. Matsuura, "A regular expression matching circuit based on a modular non-deterministic finite automaton with multi-character transition," SASIMI'10, pp.359–364, Taipei, Oct. 2010.
[15] H. Nakahara, T. Sasao, and M. Matsuura, "A regular expression matching using non-deterministic finite automaton," MEMOCODE'10, pp.73–76, Grenoble, France, July 2010.
[16] M. Pellauer, A. Agarwal, A. Khan, M.C. Ng, M. Vijayaraghavan, F. Brewer, and J. Emer, "Design contest overview: combined architecture for network stream categorization and intrusion detection (CANSCID)," MEMOCODE'10, pp.69–72, Grenoble, France, July 2010.
[17] "PCRE: Perl Compatible Regular Expression," http://www.pcre.org/.
[18] H.C. Roan, W.J. Hawang, and C.T. Dan Lo., "Shift-or circuit for efficient network intrusion detection pattern matching," FPL'06, pp.785–790, 2006.
[19] R. Sidhu and V.K. Prasanna, "Fast regular expression matching using FPGA," FCCM'01, pp.227–238, 2001.
[20] "SNORT official web site," http://www.snort.org.
[21] "SPAMASSASSIN: Open-Source Spam Filter," http://spamassassin.apache.org/
[22] "Spartan III data sheet," http://www.xilinx.com/
[23] L. Tan, and T. Sherwood, "A high throughput string matching architecture for intrusion detection and prevention," ISCA'05, pp.112–122, 2005.
[24] "Using Look-up tables as shift registers (SRL16)," http://www.xilinx.com/support/documentation/application_notes/xapp465.pdf
[25] F. Yu, Z. Chen, Y. Diao, T.V. Lakshman, and R.H. Katz, "Fast and memory-efficient regular expression matching for deep packet inspection," ANCS'06, pp.93–102, 2006.

Hiroki Nakahara received the BE, ME, 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, and IPSJ Yamashita SIG Research Award in 2011, respectively. His research interests include logic synthesis, reconfigurable architecture, and embedded systems. He is a member of the IEEE.

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 and Electronics. 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.