A Regular Expression Matching Circuit Based on a Decomposed Automaton

Hiroki Nakahara, Tsutomu Sasao, and Munehiro Matsuura

Kyushu Institute of Technology, Japan

Abstract. In this paper, we propose a regular expression matching circuit based on a decomposed automaton. To implement a regular expression matching circuit, first, we convert regular expressions into a non-deterministic finite automaton (NFA). Then, to reduce the number of states, we convert the NFA into a modular non-deterministic finite automaton with unbounded string transition (MNFAU). Next, to realize it by a feasible amount of hardware, we decompose the MNFAU into the deterministic finite automaton (DFA) and the 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.

1 Introduction

1.1 Regular Expression Matching for Network Applications

A regular expression represents a set of strings. **A regular expression matching** detects a pattern written by regular expressions from the input string. Various network applications (e.g., intrusion detection systems [15][8], a spam filter [16], a virus scanning [6], and an L7 filter [10]) use the regular expression matching. The regular expression matching spends a major part of the total computation time. Since the modern network transmission speed exceeds one Giga bit per second (Gbps), the hardware regular expression matching is essential. For the network applications, since the high-mix low-volume production and the flexible support for new protocols are required, reconfigurable devices (e.g., FPGAs) are used. Recently, dedicated high-speed transceivers for the high-speed network are embedded to FPGAs. So, a trend for using FPGAs will accelerate in the future. Although the operation speed for the regular expression matching on the FPGA exceeds one Gbps [5][3][4], these methods require high-end and high-cost FPGAs. In this paper, we realize a low-cost regular expression matching using a low-end FPGA and off-chip memory.

1.2 Related Work

Regular expressions can be detected by finite automata. In a **deterministic finite automaton (DFA)**, for each state for an input, a unique transition exists, while in a **non-deterministic finite automaton (NFA)**, for each state for an input, multiple transitions exist. In an NFA, there exists 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. [18]; a combination of the bit-partitioned DFA and the MPU [3]; and a pipelined DFA [5]. Also, various NFA-based regular expression matching 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) [14]; prefix sharing of regular expressions [11]; and a method that maps repeated parts of regular expressions to the Xilinx FPGA primitive (SRL16) [4].

1.3 Features of the Proposed Method

Lower Complexity than Existing Methods. In this paper, we compare the NFA, the DFA, and the decomposed NFA with string transition on parallel hardware model. The decomposed NFA is much smaller than conventional methods.

Efficient Utilization of Embedded Memory The conventional NFA based method uses single-character transition [14]. In the circuit, each state for the NFA is implemented by a flip-flop and an AND gate. Also, an ε-transition is realized by OR gates and routing on the FPGA. Although the modern FPGA consists of LUTs and embedded memories, the conventional NFA based method fails to use embedded memories. In contrast, our method can use both LUTs and embedded memory to implement the decomposed NFA with string (multi-character) transition.

The rest of the paper is organized as follows: Section 2 shows a regular expression matching circuit based on the finite automaton (FA); Section 3 shows a regular expression matching circuit based on an NFA with string transition; Section 4 compares complexities on the parallel hardware model; Section 5 shows the experimental results; and Section 6 concludes the paper.

2 Regular Expression Matching Circuit Based on Automaton

A regular expression consists of **characters** and **meta characters**. A character is represented by 8 bits. The **length** of the regular expression is the number of characters. Table 1 shows meta characters considered in this paper.

2.1 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 character;* δ *is a transition function (*$\delta : S \times \Sigma \rightarrow S$*);* $s_0 \in S$ *is the initial state; and* $A \subseteq S$ *is a set of accept states. In the practical network application,* $|\Sigma| = 2^8 = 256$.

Table 1. Meta Characters Used in This Paper.

Meta Character	Meaning
.	An arbitrary character
*	Repetition of more than zero or zero (Kleene closure)
+	Repetition of more than one or equal to one
?	Repetition of equal and less than one
^	Pattern to be matched at only start of the input
$	Pattern to be matched at only end of the input
()	Specify the priority of the operation
[]	Set of characters
[^]	Complement set of characters
{n,m}	Repetition (more than n and less than m)
{n,}	Repetition (more than n)
{n}	Repetition (n times)
|	Logical OR

Definition 2.2 *Let $s \in S$, and $c \in \Sigma$. If $\delta(s, c) \neq \{\phi\}$, then c denotes* **a transition character for the state** s.

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 let the extended transition function be $\hat{\delta} : S \times \Sigma^+ \rightarrow S$. 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 $C_{in} \subseteq \Sigma^+$, and $a \in A$. Then, M_{DFA} accepts a string C_{in}, if the relation holds*

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

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

Example 2.1 *Fig. 1 shows the DFA for the regular expression "A+[AB]{3}D".* ■

Example 2.2 *Consider the string matching for an input "AABAD" using the DFA shown in Fig. 1. 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.* ■

Fig. 2 shows the DFA machine, where the register stores the present state, and the memory realizes the transition function δ. Let q be the number of states, and $|\Sigma| = n$ be the number of characters in Σ. Then, the amount of memory to implement the DFA is $2^{\lceil log_2 n \rceil + \lceil log_2 q \rceil} \times \lceil log_2 q \rceil$ bits [1].

[1] 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.

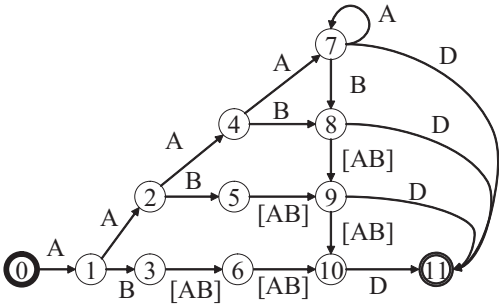

Fig. 1. DFA for the regular expression "A+[AB]{3}D".

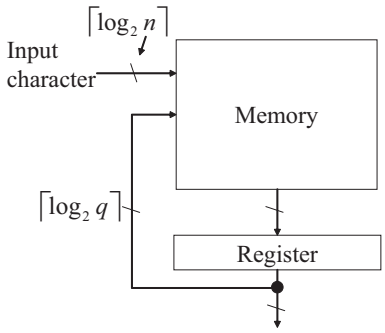

Fig. 2. DFA machine.

2.2 Regular Expression Matching Circuit Based on Non-deterministic 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 ones in Definition 2.1, while the transition function* $\gamma : S \times (\Sigma \cup \{\varepsilon\}) \rightarrow P(S)$ *is different. Note that,* ε *denotes an empty character, and* $P(S)$ *denotes a 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 ε input denotes **an** ε **transition**. In this paper, in a state transition diagram, an ε symbol with an arrow denotes the ε transition. Fig. 3 shows conversions of regular expressions into NFAs, where a gray state denotes an accept state.

Example 2.3 *Fig. 4 shows the NFA for the regular expression for "A+[AB]{3}D", and state transitions for the input string "AABAD". In Fig. 4, each element of the vector corresponds to a state of the NFA, and '1' denotes an active state.* ■

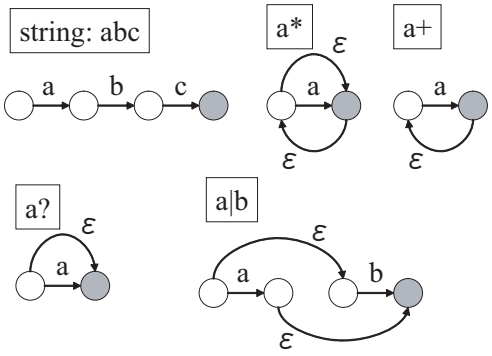

Fig. 3. Conversion of regular expression into NFA.

Sidhu and Prasanna [14] realized an NFA with single-character transitions for regular expressions [2]. Fig. 5 shows the circuit for the NFA. To realize the NFA, first, the

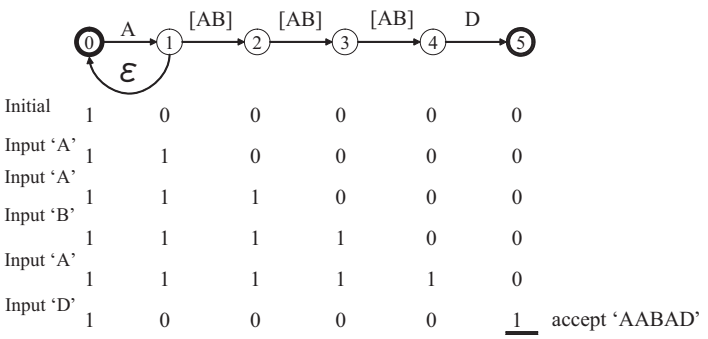

Fig. 4. NFA for the regular expression "A+[AB]{3}D".

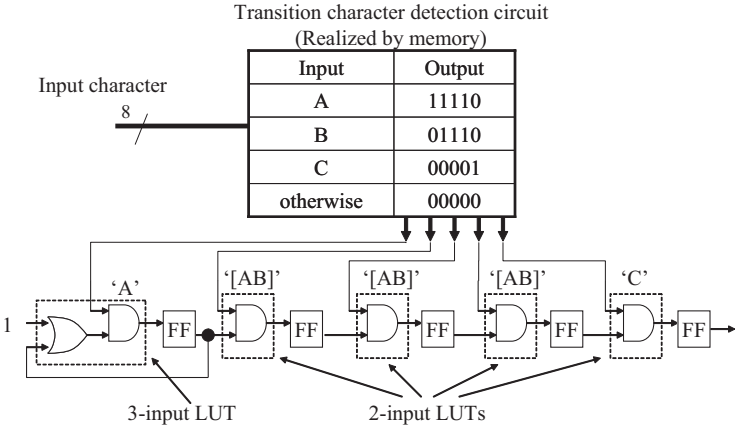

Fig. 5. A circuit for the NFA shown in Fig. 4.

memory detects the character for the state transition, and then the character detection signal is 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 routing 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 does not use the embedded memory[2]. So, their method is inefficient with respect to the resource utilization of FPGA, since modern FPGA consists

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

of LUTs and embedded memories. In the circuit for the NFA, each state is implemented by an LUT of an FPGA. Thus, the necessary number of LUTs is equal to the number of states. To reduce the number of states, we propose a regular expression matching circuit based on a **modular 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 only merge the states using the following:

Lemma 3.1 *Let $S = \{s_0, s_1, \ldots, s_{q-1}\}$ be a set of states, and $S_i \subseteq S$. Consider a partition $S = S_1 \cup S_2 \cup \cdots \cup S_u$, where $S_i \cap S_j = \phi (i \neq j)$. Let e_r be the number of ε transition inputs and outputs in the state s_r. Then, $S_i = \{s_k, s_{k+1}, \ldots, s_{k+p}\}$ can be merged into one state of the MNFAU, if $e_r = 0$ for $r = k, k+1, \ldots, k+p-1$.*

Definition 3.6 *Suppose that a set of states $\{s_k, s_{k+1}, \ldots, s_{k+p}\}$ be merged into a state S_M of MNFAU. A string $C = c_k c_{k+1} \cdots c_{k+p}$ is* **a transition string of** *S_M, when $c_j \in \Sigma$ is a transition character of s_j for $j = k, k+1, \ldots, k+p$.*

Example 3.4 *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, and its transition string is "[AB][AB][AB]D". However, the set of states $\{s_1, s_2\}$ cannot be merged, since $e_1 \neq 0$.* ■

Example 3.5 *Fig. 6 shows the MNFAU derived from the NFA shown in Fig. 4.* ■

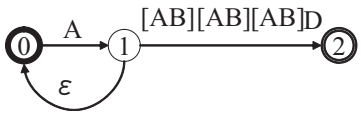

Fig. 6. MNFAU derived from the NFA shown in Fig. 4.

3.2 Realization of MNFAU

To realize the MNFAU, as shown in Fig. 7, we consider **a decomposed MNFAU**, and realize it by the following circuits:

1. The transition string detection circuit
2. The state transition circuit.

Since transition strings do not include meta characters [3], they are detected by **the exact matching**. The exact matching is a subclass of the regular expression matching and the DFA can be realized by feasible amount of hardware [20]. On the other hand, since the state transition part handles the ε transition, it is implemented by the cascade of logic cells shown in Fig. 5.

[3] However, a set of characters "[]" can be used.

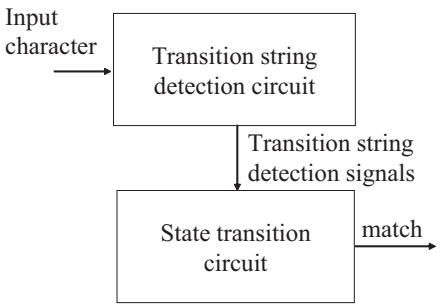

Fig. 7. Decomposition of the MNFAU.

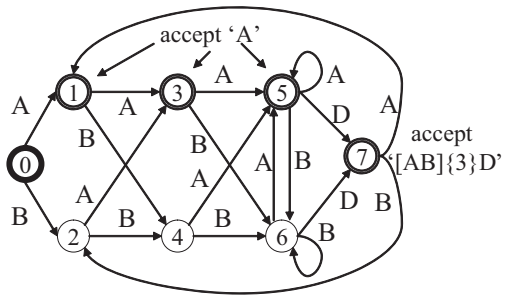

Fig. 8. AC-DFA accepting strings "A" and "[AB][AB][AB]D".

Transition String Detection Circuit. Since each state of the MNFAU can merge different number of states of the NFA, the lengths of the transition strings for states of the MNFAU can be also 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. 2.

Example 3.6 *Fig. 8 illustrate the AC-DFA accepting transition strings "A" and "[AB][AB][AB]D" for the MNFAU shown in Fig. 6.* ■

State Transition Circuit [13]. Fig. 9 shows the state transition circuit for the MNFAU. When the AC-DFA detects the transition string ("ABD" in Fig. 9), the detection signal is sent to the state transition circuit. Then, the state 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 shift register is inserted between two states in the MNFAU to synchronize with the AC-DFA. A 4-input LUT of a Xilinx FPGA can also be used as a shift register (SRL16) [19]. Fig. 10 shows two LUT modes of a Xilinx FPGA.

Fig. 11 shows the 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, u-bit detection signal is necessary for the state transition circuit, where u is the number of states for the MNFAU. We use the decoder that converts $\lceil log_2 q \rceil$-bit state to u-bit detection signal. Since the decoder is relatively small, it is implemented by the embedded memory in the FPGA.

Example 3.7 *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.* ■

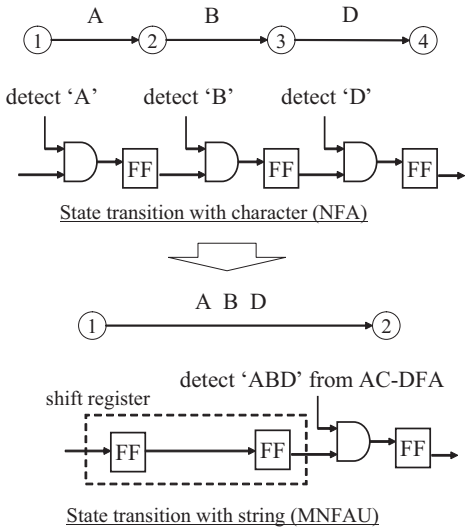

Fig. 9. State transition with the string.

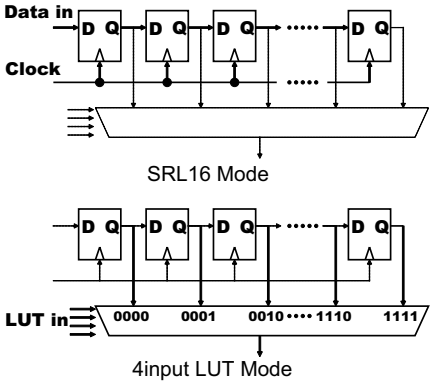

Fig. 10. Two LUT modes for Xilinx FPGA.

4 Complexity of Regular Expression Matching Circuit on Parallel Hardware Model

For the Xilinx FPGA, **a logic cell (LC)** consists of a 4-input look-up table (LUT) and a flip-flop (FF) [17]. Also, the FPGA has embedded memories. Therefore, as for the area complexity, we consider both **the LC complexity** and **the embedded memory complexity**.

4.1 Theoretical Analysis

Aho-Corasick DFA. As shown in Fig. 2, the machine for the DFA has the 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. [20] showed that, for m regular expressions with length s, the memory complexity is $O(|\Sigma|^{sm})$, where $|\Sigma|$ denotes the number of characters in Σ.

Baeza-Yates NFA. As shown in Fig. 5, the NFA consists of the memory for the transition character detection, and the cascade of small machine consisting 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 $O(ms)$. The regular expression matching circuit based on the NFA has s states and performs ε transitions at a time in every clock. By using m parallel circuits shown in Fig. 5, the circuit can match m regular expressions in parallel. Thus, the time complexity is $O(1)$.

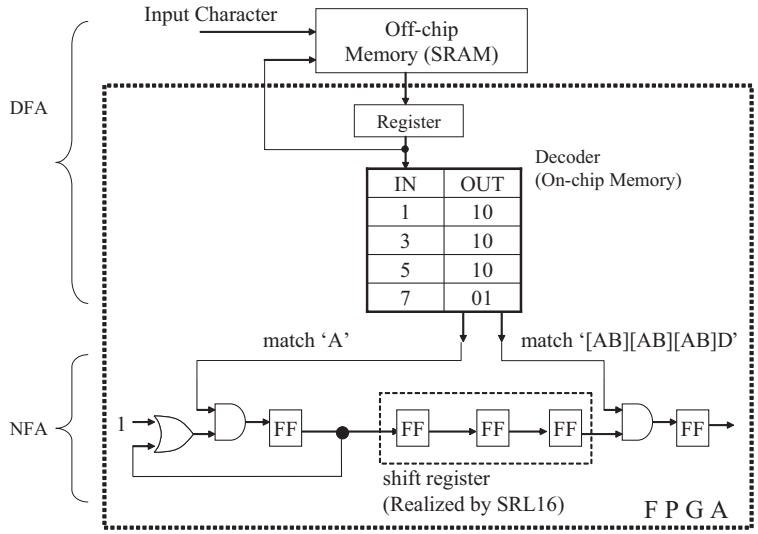

Fig. 11. An example of the circuit for the MNFAU.

Decomposed MNFAU. As shown in Fig. 7, the decomposed MNFAU consists of the transition string detection circuit and the state transition circuit. The transition string detection circuit is realized by the DFA machine shown in Fig. 2. Let p_{max} be the maximum length of the regular expression, and $|\Sigma|$ be the number of characters in a set of 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 amount of memory for the MNFAU is larger than that for the NFA. The memory for the NFA requires sm-bit words, while the memory for the MNFAU requires $\lceil log_2 q \rceil$-bit words [4]. For the NFA, off-chip memories are hard to use, since the number of pins on the FPGA is limited. Thus, the NFA requires a large number of on-chip memories. On the other hand, the MNFAU can use off-chip memory, since the required number of pins is small. From the point of the implementation, although the MNFAU requires larger amount of memory than the NFA, the MNFAU can use off-chip memory and a small FPGA.

Table 2 compares the area and time complexities for the DFA, the NFA, 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, while the number of LCs is reduced to $\frac{1}{p_{max}}$ of the NFA.

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

4.2 Analysis Using SNORT

To verify these analyses, 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 [15], 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. Fig. 12 shows the relation of the length of the regular expression s and the number of LCs, while Fig. 13 shows the relation of s and the memory size. Note that, in Fig. 12, the vertical axis is a linear scale, while, in Fig. 13, it is a logarithmic scale. As shown in Fig. 12, the ratio between the number of LCs and s is the constant. On the other hand, as shown in Fig. 13, 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.

Table 2. Complexities for the DFA, the NFA, 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	$O(1)$	$O(\|\Sigma\|^{p_{max}})$	$O(\frac{ms}{p_{ave}})$

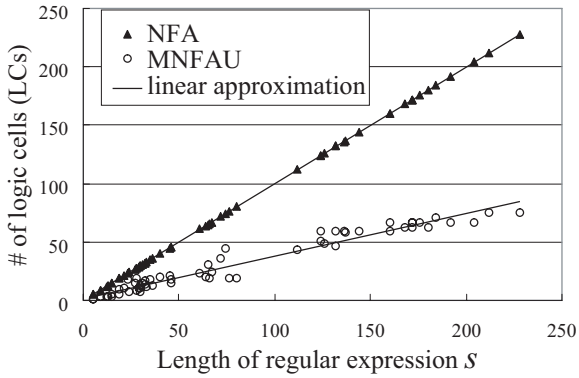

Fig. 12. Relation the length s of regular expression and the number of LCs.

5 FPGA Implementation

We selected the regular expressions from SNORT (open-source intrusion detection system), and generated the decomposed MNFAU. Then, we implemented to the Xilinx Spartan III FPGA (XC3S4000: 62,208 logic cells (LCs), total 1,728 Kbits BRAM). The

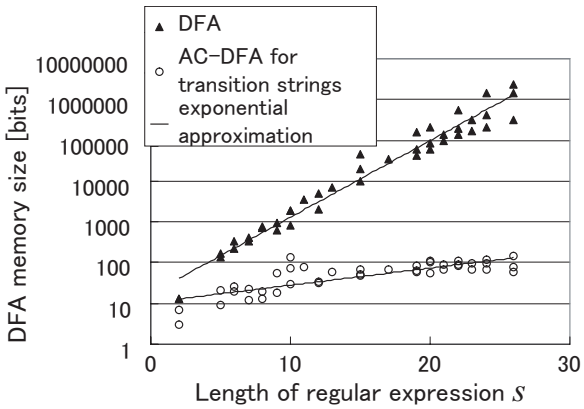

Fig. 13. Relation the length s of regular expression and the memory size.

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) [12] (SASIMI'10)	MNFA(p)	Virtex 6	3.2	4,717	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

total number of rules 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-Mbit off-chip SRAM is used to store the transition function of the AC-DFA, while 1,585-Kbit 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.

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 is 1.56-5.12 times smaller than the NFA method.

6 Conclusion

In this paper, we proposed a regular expression matching circuit based on a decomposed MNFAU. To implement the circuit, first, we convert the regular expressions to an NFA. Then, to reduce the number of states, we convert the NFA into the MNFAU. Next, to realize it by a feasible amount of the hardware, we decompose the MNFAU into the transition string detection part and the state transition part. The transition string detection part is implemented by an off-chip memory and a simple sequencer, while the state transition part is implemented by a cascade of logic cells. Also, this paper shows that the MNFAU based implementation has lower area complexity than the DFA and the NFA based ones. The implementation of 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 per a character, the MNFAU is 1.56-5.12 times smaller than NFA methods.

7 Acknowledgments

This research is supported in part by the grant of Regional Innovation Cluster Program (Global Type, 2nd Stage).

References

1. A. V. Aho, and M. J. Corasick, "Efficient string matching: An aid to bibliographic search," *Comm. of the ACM*, Vol. 18, No. 6, pp. 333-340, 1975.
2. R. Baeza-Yates, and G. H. Gonnet, "A new approach to text searching," *Communications of the 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. Z. Kohavi, *Switching and Finite Automata Theory, McGraw-Hill Inc.*, 1979.
10. "Application Layer Packet Classifier for Linux," http://l7-filter.sourceforge.net/
11. C. Lin, C. Huang, C. Jiang, and S. Chang, "Optimization of regular expression pattern matching circuits on FPGA," *DATE'06*, pp.12-17, 2006.
12. 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*, Taipei, Oct. 18-19, 2010, pp. 359-364.
13. H. Nakahara, T. Sasao, and M. Matsuura, "A regular expression matching using non-deterministic finite automaton," *MEMOCODE'10*, Grenoble, France, July 26-28, 2010, pp. 73-76.
14. R. Sidhu, and V. K. Prasanna, "Fast regular expression matching using FPGA," *FCCM'01*, pp. 227-238, 2001.
15. "SNORT official web site," http://www.snort.org.
16. "SPAMASSASSIN: Open-Source Spam Filter," http://spamassassin.apache.org/
17. "Spartan III data sheet," http://www.xilinx.com/
18. L. Tan, and T. Sherwood, "A high throughput string matching architecture for intrusion detection and prevention," *ISCA'05*, pp.112-122, 2005.
19. "Using Look-up tables as shift registers (SRL16)," http://www.xilinx.com/support/documentation/application_notes/xapp465.pdf
20. 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.