A Regular Expression Matching Circuit Based on a Modular Non-Deterministic Finite Automaton with Multi-Character Transition

Hiroki Nakahara Tsutomu Sasao Munehiro Matsuura

Kyushu Institute of Technology, Department of Computer Science and Electronics,
680–4, Kawazu, Iizuka, Fukuoka, 820–8502 Japan

Abstract— This paper shows an implementation of a regular expression circuit based on an NFA (Non-deterministic finite automaton). Also, it shows that the NFA based one is superior to the DFA (Deterministic finite automaton) based one, in terms of area and time complexity. A regular expression matching circuit is generated as follows: First, the given regular expressions are converted into an NFA. Then, to reduce the number of states, the NFA is converted into a modular non-deterministic finite automaton (MNFA(p)) with p-character transition. Finally, a finite-input memory machine (FIMM) to detect p-characters as well as the matching elements (MEs) realizing the states for the MNFA(p) are generated. We designed MNFA(p) for different p on a Xilinx FPGA. Then, we derived an optimal value p that efficiently uses both LUTs and embedded memories of the FPGA. As for the performance per FPGA area, our method is 6.2-18.6 times better than DFA-based methods, and is 1.8 times better than the NFA-based method. Since our method efficiently utilizes FPGA resource, a low-cost FPGA can be used to implement a high-performance regular expression matching circuit.

I. Introduction

A. Intrusion Detection System

An intrusion detection system (IDS) monitors network against malicious activities or policy violations to alert network administrators. The IDS is roughly classified into two: a network IDS and a host IDS. To perform intrusion detection, the network IDS [1] compares present network activities with normal ones. On the other hand, the host IDS performs **pattern matching** to see if incoming data matches known malicious activities (**patterns**) represented by **regular expressions**. Thus, the host IDS is called a pattern matching based IDS. Open source IDS exist such as SNORT [14] for a server, and Firekeeper[7] for a personal browser (FireFox). Both open source IDSs are based on regular expression matching. Many network security vendors adopt pattern matching based IDS, since the pattern matching-based one is faster and easier to implement than the network IDS. Recently, **an IPS (Intrusion Prevention System)** consisting of the IDS and the firewall come into practical use. The IPS blocks the connection of suspicious activity. The bottleneck of the host IDS is the regular expression matching.

B. Related Works

The 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, a multiple transitions exists. In an NFA, there exist **ε-transitions** to other states without consuming input characters. Various DFA-based regular expression matching exist; an Aho-Corasick algorithm [1]; a bit-partition of the Aho-Corasick DFA by Tan et al. [16]; 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 (Prasanna's method) [13]; prefix sharing of regular expressions [9]; and a method that maps repeated parts of regular expressions to the Xilinx FPGA primitive (SRL16) [4].

C. Contributions of This Paper

Comparison the parallel hardware for the NFA with that for the DFA Yu et al. compared complexities of the NFA with the DFA on random access machine (RAM) model [18]. In this paper, we compare NFA and DFA on FPGA circuits.

Regular expression matching circuit for the NFA with multi-character transition[2] Prasanna et al. implemented a regular expression matching circuit by an NFA with single-character transition [13]. In their machine, each state for the NFA was implemented by a single-character detector and an AND gate. Also, an ε-transition was realized by OR gates and routing on the FPGA. Although the modern FPGA consists of LUTs and

[1]An anomaly-based IDS.

[2]"multi-character transition" has been defined in [12].

embedded memories, Prasanna's method failed to utilize embedded memories. So, in their implementation, embedded memories were left unused. In contrast, our method utilizes both LUTs and embedded memory to implement an NFA with p-character transition.

The rest of the paper is organized as follows: Chapter 2 shows the implementation of the regular expression matching circuit based on the NFA; Chapter 3 compares parallel hardware for the NFA with that for the DFA; Chapter 4 explains the regular expression matching circuit for the NFA with multi-character transition; Chapter 5 shows the experimental results; and Chapter 6 concludes the paper.

II. Regular Expression Matching Based on NFA

A. Regular Expression

A regular expression consists of **characters** and **meta characters**. Our implementation accepts the following meta characters: '*' (repetition of more than zero character); '?' (zero or one character); '.' (an arbitrary character); '+' (more than one repetition of character); '()' (specify the priority of the operation); '|' (logical OR).

B. Regular Expression Matching Circuit Based on NFA

Fig. 1 illustrates conversions of regular expressions into NFAs, where 'ε' denotes an ε-transition, and a gray state denotes an accept state. Fig. 2 shows an NFA accepting the regular expression 'abc(ab)*a', and state transitions with the input string 'abca'. In Fig. 2, each element of the vector corresponds to a state of the NFA, and '1' denotes an active state. Fig. 3 shows the circuit realizing the NFA in Fig. 2. To realize the NFA, first, the memory detects the character for the state transition, and then it sends the character detection signal to the **matching element (ME)**. Each ME corresponds to a state of the NFA, and the ME for the accepted state generates the match signal. In Fig. 3, in each ME, the *FF* corresponds to the element of the vector shown in Fig. 2; i denotes the matching signal from the previous state; o denotes the matching signal to the next state; c denotes the character detection signal; *ei (eo)* denotes the input (output) signal for the ε-transition.

III. Complexities of Parallel Hardware for NFA and DFA

In this section, we compare the parallel hardware for NFA with that for DFA. We define **the length of the regular expression** s as the number of non-meta characters in the regular expression. We show that the NFA-based parallel hardware is superior to the DFA-based one with respect to the area complexity.

Example 3.1 *For a regular expression "abc(ab)*a" shown in Fig. 2, the length of the regular expression is* $s = 6$. *The length of the regular expression is not always equal to the number of states for the NFA.* ■

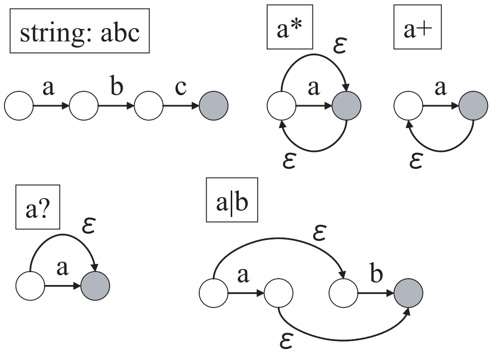
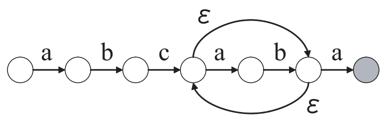


Fig. 1. Conversions of Regular Expressions into NFAs.



```
initial         (1, 0, 0, 0, 0, 0, 0)
input 'a'       (1, 1, 0, 0, 0, 0, 0)
ε -transition   (1, 1, 0, 0, 0, 0, 0)
input 'b'       (1, 0, 1, 0, 0, 0, 0)
ε -transition   (1, 0, 1, 0, 0, 0, 0)
input 'c'       (1, 0, 0, 1, 0, 0, 0)
ε -transition   (1, 0, 0, 1, 0, 1, 0)
input 'a'       (1, 1, 0, 0, 1, 0, 1) Accept 'abca'
```

Fig. 2. An Example of NFA.

A. Complexity for Regular Expression Matching Circuit Based on NFA

As shown in Fig. 3, an ME consists of an OR gate, an AND gate, and a flip-flop. Thus, the ME can be implemented by 3-inputs LUT and a flip-flop. Note that, a modern Xilinx FPGA has logic cells that consist of 4-input LUTs and flip-flops. For m regular expression with length s, the number of LUTs is $O(ms)$. On the other hand, from Fig. 3, the one character (8 bits) detector consists of an 8-input memory. Since the amount of memory is $m \times 2^8 \times s$ bits, the memory size is also $O(ms)$.

The parallel hardware based on NFA shown in Fig. 3 can activate s states and s ε transitions per one clock. Since m regular expressions are realized by m parallel hardware, the time complexity is $O(1)$.

We selected regular expressions from open-source IDS SNORT [14]. Then, we implemented the parallel hardware based on NFA shown in Fig. 3. Next, we obtained the necessary number of LUTs and the memory size. Fig. 4 shows the relation of the length of the regular expression s and the number of LUTs. Fig. 5 shows the relation of s and the memory size. These figures show that the amount of hardware increases linearly with s.

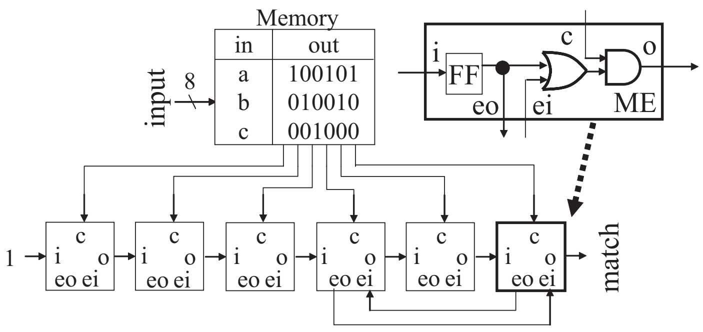


Fig. 3. A Circuit Realizing NFA [13].

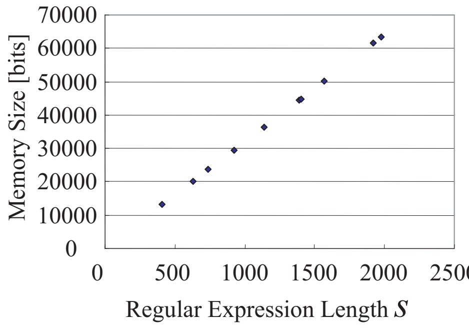


Fig. 5. Relation the Length s of Regular Expression and the Memory Size.

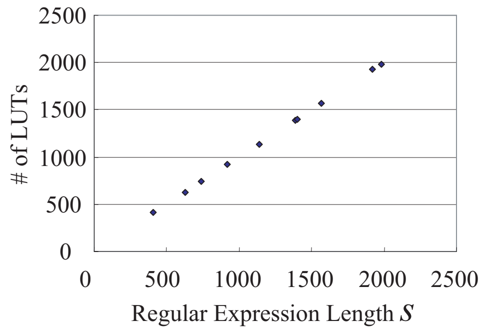


Fig. 4. Relation the Length s of Regular Expression and the Number of LUTs.

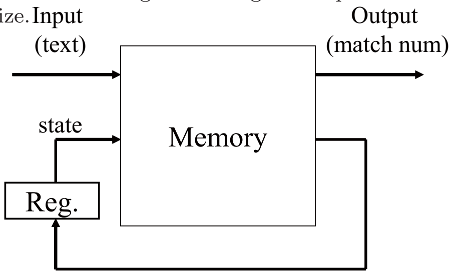


Fig. 6. A Sequencer Emulating DFA.

B. Comparison with DFA

Fig. 6 shows a sequencer that emulates the DFA, where the memory stores the next states, and the register stores the present state. In the sequencer, since state transition is performed in every clock, the time complexity is $O(1)$. On the other hand, for the area of the sequencer, the memory occupies the almost all part of the area. Since the hardware other than the memory is considered to be constant, the number of LUTs for sequencer is $O(1)$.

Yu et al. analyzed that the memory size for the DFA is $O(|\sum|^{nm})$, where $|\sum|$ denotes the number of input symbols [18]. For standard regular expressions, $|\sum| = 2^8 = 256$. Tan et al. considered the bit-partitioned sequencer for the DFA [16]. Sherwood et al. showed that the number of states for the bit-partitioned DFA does not exceed that for the original DFA [6]. Unfortunately, even if the bit-partitioned DFA can reduce the memory size, the memory size remains $O(|\sum|^{sm})$. Since the number of patterns represented by regular expressions is expected to increase drastically, for the DFA-based hardware, the explosion of the memory size will be a crucial problem. Table I compares complexities for the NFA and the DFA.

IV. Compaction of NFA-based Regular Expression Circuits

Sidhu and Prasanna [13] implemented a regular expression matching circuit based on an NFA with single-character transition [2]. Each state for the NFA was implemented by a single character detector and an AND gate. Also, an ε-transition was realized by OR gates and routing on the FPGA. Although modern FPGAs consist of LUTs and embedded memories, their method failed to utilize embedded memories[3]. So, their method is inefficient with respect to the resource utilization of FPGA. In contrast, our method implements an NFA with p-character transition by embedded memories and LUTs to utilize the FPGA resources efficiently.

A. NFA with Multi-Character Transition [10]

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 use the **NFA with p-character transition modular non-deterministic finite automaton: MNFA(p)**. Note that, an MNFA(1) corresponds to an ordinary NFA and is simply denoted by 'NFA'. To convert an NFA into an MNFA(p), we concatenate characters for sequence of the states. However, to retain the ε-transition, only the nodes without ε-transition edge are merged to the parent node. Fig. 7 shows the MNFA(3) that is derived from the NFA shown in Fig. 2.

TABLE I
Complexities for NFA-based and DFA-based Circuits on FPGA.

		Bit-partitioned DFA	Prasanna-NFA
Area	# LUT	$O(1)$	$O(ms)$
	Memory	$O(\vert\sum\vert^{ms})$	$O(ms)$
Time		$O(1)$	$O(1)$

[3]They used LUTs to implement single-character detectors (comparators) instead of the memory shown in Fig. 6

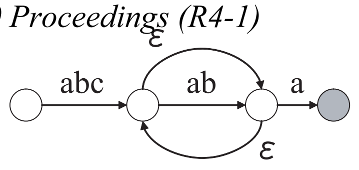


Fig. 7. MNFA(3) Equivalent to NFA Shown in Fig. 2.

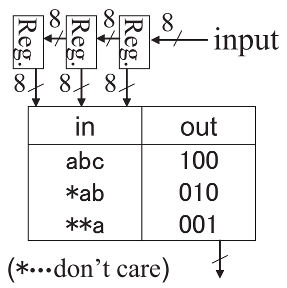


Fig. 8. Finite Input Memory Machine (FIMM).

B. Regular Expression Matching Circuit in MNFA(p)-based Circuit

In an MNFA(p), for each state, there exist transition to other states by consuming a string with up to p characters. For the MNFA(3) shown in Fig. 7, the set of transition strings is {abc,ab,a}. To detect the transition strings, we use the **finite input memory machine (FIMM)**. Fig. 8 illustrates the FIMM that detects the strings {abc,ab,a}. When the FIMM detects a string, it generates a detection signal. Fig. 9 illustrates the circuit for three-character transition. To synchronize the detection signal from the FIMM and the matching signal from the preceding ME, we insert shift registers. An LUT of a Xilinx FPGA can also be used as a shift register (SRL16) [17]. Fig. 10 shows a 4-input LUT of a Xilinx FPGA. Let p be the maximum number of characters for the transition strings of the MNFA(p). The single-memory realization of the FIMM requires $p2^{8p}$ bits. In Fig. 11, since $p = 3$, the necessary memory size is 48 mega bits, which is impractical. Our method decomposes the memory of the FIMM into p parts and uses the bitwise-AND [11]. Each part of the memory is implemented by an embedded memory of the FPGA.

Example 4.2 *Fig. 12 shows the circuit for the MNFA(3) in Fig. 9. To realize the multi-character transition, we insert shift registers into MEs. When the FIMM detects a transition string, it sends the detection signals to the corresponding ME. Then, the ME performs the multi-character transition.* ■

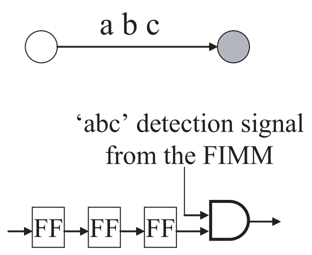


Fig. 9. Circuit for Multi-Character Transition.

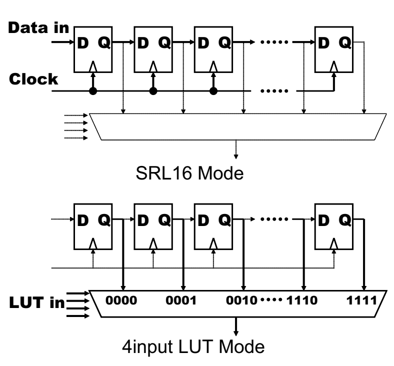


Fig. 10. 4-input LUT of a Xilinx FPGA.

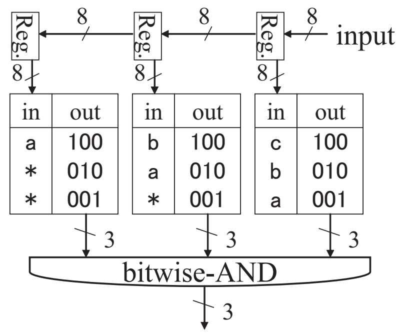


Fig. 11. Partition of FIMM.

C. Partition of Memory for FIMM

In Fig. 11, assume that the FIMM consists of memories with 8 inputs and 3 outputs. We can further reduce the total amount of memory by partitioning the inputs of the memory. In this part, we consider the optimal partition of the inputs. In the standard regular expression, one

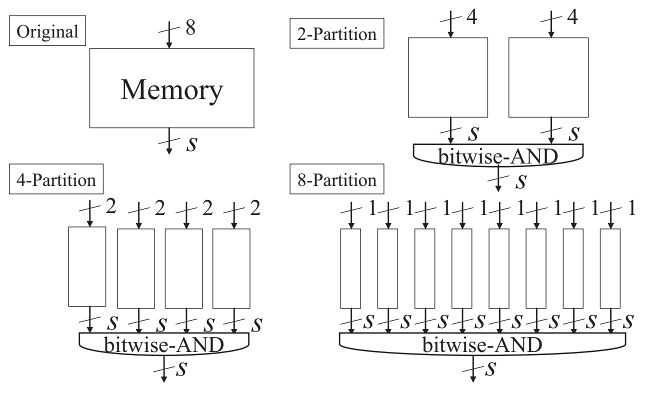


Fig. 13. Four Different Partitions of Memory for the FIMM.

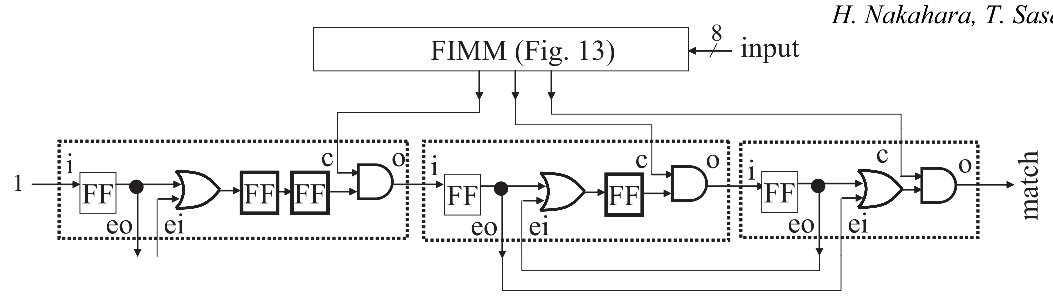


Fig. 12. Circuit for MNFA(3).

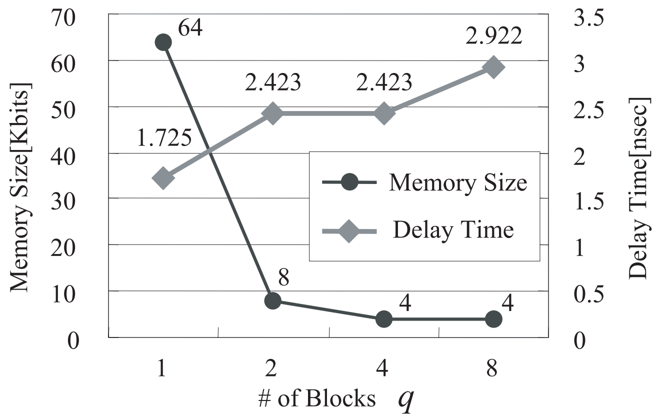


Fig. 14. Trade-Off of Memory Size and Delay Time.

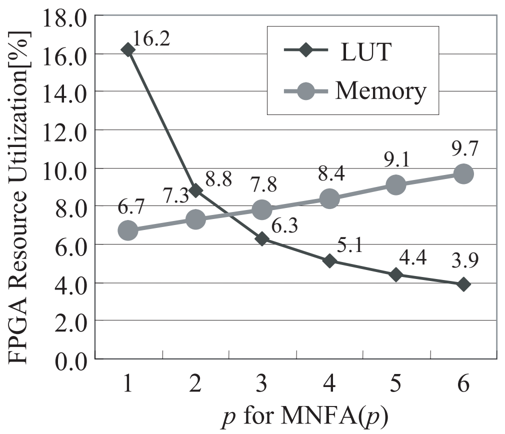


Fig. 15. Resource Utilization of FPGA.

character is 8 bits. Thus, the memory is partitioned into four (Fig. 13). Let q be the number of blocks for the partition of 8-input s-output memory. Then, the total amount of memory $M_{FIMM}(q)$ is

$$M_{FIMM}(q) \quad = \quad q \times 2^{\frac{8}{q}} \times s. \qquad (1)$$

Although we can reduce the amount of memory by increasing q, the large bitwise-AND is necessary. Thus, the trade-off of the memory size and the delay time for the bitwise-AND exits among different q. We implemented four different partitions of the memory shown in Fig. 13 to a Xilinx FPGA (Virtex 6) to obtain the memory size and the delay time, where we assume that $s = 256$. Fig. 14 shows the memory size and the delay time with respect to the number of blocks. Memory sizes for $M_{FIMM}(4)$ and $M_{FIMM}(8)$ are the same. For $q = 2$, the bitwise-AND part is implemented by s 2-input AND gates, while for $q = 4$, that is implemented by s 4-input AND gates. The Xilinx FPGA has 4-inputs LUTs that can realize both 2-input and 4-input AND gates by one LUT. So, delay times for $q = 2$ and $q = 4$ are the same. From Fig. 14, for $q = 4$, the memory size and the delay time keep their balance. However, for practical implementation of the Xilinx FPGA, we chose $q = 2$. This is because the Xilinx FPGA contains the embedded memory (BRAM) that is 8 Kbits dual port memory. For $q = 4$, even if we use dual port BRAM, two BRAMs are necessary, and it's utilization is a half of the case $q = 2$. On the other hand, for $q = 2$, we can implement the memory part of the FIMM by a dual port BRAM.

V. Experimental Results

A. *Optimal p for MNFA(p)*

For the regular expression matching circuit based on MNFA(p), let q be the number of blocks for the memory for the FIMM; s be the regular expression length; and $s(p)$ be the number of states. Then, the total amount of memory $M(p)$ is

$$M(p) \quad = \quad q \times 2^{\frac{8}{q}} \times p \times s(p). \qquad (2)$$

From the preliminary experiment, we chose $q = 2$. When the MNFA(p) has no ε-transition, the doubling value p will decrease the number of states $s(p)$ into a half. In this case, the number of address of the memory of FIMM $q \cdot 2^{\frac{8}{q}} \cdot p$ is doubled. Therefore, such change of the value of p does not change the total amount of memory $M(p)$.

However, actually, an MNFA(p) has ε-transition. Even if we double the value of p, $s(p)$ does not decrease to a half. From this observation, we have $\frac{1}{2}s(p) \leq s(2p)$. Thus, it implies the relation $M(p) \leq M(2p)$. Thus, for MNFA(p), the total amount of memory of the FIMM increases with p, while the number of states for the MNFA(p) decreases with p.

To obtain an optimal p experimentally, we implemented regular expression matching circuit based on MNFA(p) for different value of p. We selected the regular expressions from open-source IDS SNORT. Then, we implemented circuit for the regular expressions to the Xilinx FPGA (Virtex 6: XC6VLX75T, # of LUTs: 74,496, memory size: 5,616 Kbits). Fig. 15 shows the resource utilization of LUTs and memories. From Fig. 15, when $p = 2$ or $p = 3$, the ratio of resource utilizations of LUTs and memories keep their balance.

TABLE II
Comparison with Other Methods.

	Method	FPGA	Th (Gbps)	#LC	Memory (Kbits)	#Char	PEM
Brodie (ISCA'06) [5]	Pipelined-DFA	Virtex 2	4.0	247,000	3,456	11,126	0.66
Baker (FPL'06) [3]	MPU + Bit-Partitioned DFA	Virtex 4	1.4	N/A	6,000	16,715	0.22
Vassiliadis (FPT'06) [4]	Improvement of Prasanna-NFA	Virtex 4	2.9	25,074	0	19,580	2.27
Proposed Method	MNFA(3)	Virtex 6	3.2	4,707	441	12,095	4.11

B. Comparison with Other Methods

We implemented the regular expression matching circuit based on MNFA(3). The maximum clock frequency is 400 MHz. Our MNFA(3) circuit processes one character (8 bits) per one clock, so the system throughput is 3.2 Gbps. To compare with other methods, we use PEM (Performance Efficiency Metrics) proposed by Vassiliadis et al. [4]. Since 12 Bytes of memory occupies the FPGA area equivalent to a logic cell on Xilinx FPGA [15], PEM is defined by

$$PEM = \frac{Throughput\ (Gbps)}{Area\_per\_character} \quad (3)$$

$$= \frac{Throughput\ (Gbps)}{\frac{\#\ LogicCells + \frac{MemorySize\ (Bytes)}{12}}{\#\ Characters}}. \quad (4)$$

Table II compares with other methods with respect to PFM. From Table II, as for PEM, our method is 6.2-18.6 times better than DFA methods, and is 1.8 times better than the NFA method.

VI. Conclusion

This paper showed the realization of the regular expression matching circuit based on the MNFA(p). Also, this paper showed that complexity of the parallel hardware for the NFA based one is lower than that for the DFA based one. We obtained an optimal value p for the MNFA(p) experimentally. When $p = 2$ or $p = 3$, it keeps balance for the memory size and the number of LUTs. As for the PEM (Performance Efficient Metric), our method is 6.2-18.6 times better than DFA-based methods, and is 1.8 times better than the NFA-based method. The advantage of our method is efficiently of implementation, so we can realize a high-performance regular expression matching circuit with a low-end FPGA.

VII. 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," *16-th Int. Conf. on Field Programmable Logic and Applications (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," *Proc. IEEE International conference on Field Programmable Technology (FPT 2006)*, 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," *Proc. 33rd Int'l Symp. on Computer Architecture (ISCA 2006)*, pp. 191-202, 2006.

[6] R. Dixon, O. Egecioglu, and T. Sherwood, "Automata-theoretic analysis of bit-split languages for packet scanning," *Proc. 13th Int'l conf. on Implementation and Application of Automata (CIAA 2008)*, pp.141-150, 2008.

[7] "Firekeeper: Detect and block malicious sites," http://firekeeper.mozdev.org/

[8] Z. Kohavi, *Switching and Finite Automata Theory, McGraw-Hill Inc.*, 1979.

[9] C. Lin, C. Huang, C. Jiang, and S. Chang, "Optimization of regular expression pattern matching circuits on FPGA," *Proc. of the Conference on Design, Automation and Test in Europe (DATE 2006)*, pp.12-17, 2006.

[10] H. Nakahara, T. Sasao, and M. Matsuura, "A regular expression matching using non-deterministic finite automaton," *Proc. of Eighth ACM/IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE 2010)*, Grenoble, France, pp.73-76, July 26-28, 2010.

[11] H. Nakahara, T. Sasao, M. Matsuura, and Y. Kawamura, "A virus scanning engine using a parallel finite-input memory machines and MPUs," *Proc. Int'l Conf. on Field Programmable Logic and Applications (FPL 2009)* Aug. 31 - Sept. 2, 2009.

[12] A. Motoki, "Finite automaton generating device, pattern matching, device method for generating finite automaton circuit, and program," *Patent*, PCT/JP2009/060985, June, 17, 2009.

[13] R. Sidhu, and V. K. Prasanna, "Fast regular expression matching using FPGA," *Proc. of the 9th Annual IEEE symp. on Field-programmable Custom Computing Machines (FCCM 2001)*, pp. 227-238, 2001.

[14] "SNORT official web site," http://www.snort.org.

[15] T. Sproull, G. Brebner, and C. Neely, "Mutable codesign for embedded protocol processing," *Proc. of 15th Int'l Conf. on Field Programmable Logic and Applications*, 2005.

[16] L. Tan, and T. Sherwood, "A high throughput string matching architecture for intrusion detection and prevention," *Proc. 32nd Int'l Symp. on Computer Architecture (ISCA 2005)*, pp.112-122, 2005.

[17] "Using Look-up tables as shift registers (SRL16)," http://www.xilinx.com/support/documentation/application_notes/xapp465.pdf

[18] F. Yu, Z. Chen, Y. Diao, T. V. Lakshman, and R. H. Katz, "Fast and memory-efficient regular expression matching for deep packet inspection," *Proc. of the 2006 ACM/IEEE symp. on Architecture for Networking and Communications Systems (ANCS 2006)*, pp. 93-102, 2006.