Journal of Embedded Computing 3 (2007/2009) 99–107
DOI 10.3233/JEC-2009-0083
IOS Press

FPGA-based ROM-free network intrusion detection using shift-OR circuit

Wen-Jyi Hwang[a,*], Huang-Chun Roan[a], Ying-Nan Shih[a], Chia-Tien Dan Lo[b] and Chien-Min Ou[c]
[a]*Department of Computer Science and Information Engineering, National Taiwan Normal University, Taipei, 117, Taiwan*
[b]*Department of Computer Science, University of Texas at San Antonio, San Antonio, TX 78249, USA*
[c]*Department of Electronics Engineering, Ching Yun University, Chungli, 320, Taiwan*

Abstract. This paper introduces a novel FPGA-based signature match co-processor that can serve as the core of a hardware-based network intrusion detection system (NIDS). The co-processor is based on simple shift registers and bitmap encoders for the efficient signature match in hardware. As compared with related work, experimental results show that the proposed work achieves higher throughput and less hardware resource in the FPGA implementations of NIDS systems.

Keywords: Network intrusion detection system, FPGA implementation, pattern matching, Shift-Or algorithm, string searching.

1. Introduction

Due to increasing number of network worms and virus, network users are vulnerable to malicious attacks. A network intrusion detection system (NIDS) provides an effective security solution to the network attacks. It monitors network traffic for suspicious data patterns and activities, and informs system administrators when malicious traffic is detected so that proper actions may be taken. Many NIDSs such as SNORT [11] prevent computer networks from attacks using pattern-matching rules. The computational complexity of NIDSs therefore may be high because of the requirement of the string matching during their detection processes.

The software-based SNORT systems may only achieve up to 60 Mbps [7] throughput because of the high computational complexity. Since these systems do not operate at line speed, some malicious traffic can be dropped and thus may not be detected. To accelerate the speed for intrusion detection, several FPGA-based approaches have been proposed [4,5,7,9,10]. FPGA-based reconfigurable hardware can be programmed almost like software, maintaining the most attractive advantage of flexibility with less cost than traditional ASIC hardware implementation. Moreover, the FPGA hardware implementation can exploit parallelism for string matching so that the throughput of NIDSs can be increased.

The approach based on regular expressions [5,6] has been found to be effective for the FPGA implementation of SNORT rules. It results in designs with low area cost and moderate throughput acceleration. A regular expression is generated for every pattern in this approach. A nondeterministic finite automata (NFA) or a deterministic finite automata (DFA) is then used to implement each regular expression. In the finite automata implementations, efficient exploitation of parallelism is difficult because the input stream is scanned one character at a time. Another alternative for FPGA implementation is to use the content addressable memory (CAM) [4,10]. By the employment of multiple comparators in the CAM, the processing of multiple input characters per cycle is possible. This may effectively increase the throughput at the expense of higher area cost.

*Corresponding author: Department of Computer Science and Information Engineering, National Taiwan Normal University, No. 88, Sec.4, Ting-Chow Rd, Taipei, 117, Taiwan. Tel.: +886 2 2932 2411 201; Fax: +886 2 2932 2378; E-mail: whwang@ntnu.edu.tw.

ISSN 1740-4460/07-09/$17.00

Our goal is to present a novel FPGA implementation approach for NIDSs achieving both high throughput and low area cost. The proposed architecture is based on the shift-or algorithm for exact string matching [1]. The shift-or algorithm is an effective software approach for pattern matching because of its simplicity and flexibility. However, it may not perform well when the pattern size is larger than the computer word size, which is the case for many SNORT patterns. Accordingly, the software implementation of shift-or algorithm may not be suited for SNORT systems.

The proposed architecture uses only simple shift registers and bitmap encoders for the hardware implementation of shift-or algorithm. It imposes no limitation on the pattern size. In the architecture, each SNORT pattern is only associated with a shift register for pattern comparison, which are designed in accordance with the pattern size. In addition, different rules may also share the same bitmap encoder to reduce the area cost for FPGA implementation.

Because of its simplicity, the architecture may operate at a higher clock rate as compared with other implementations. The area cost may also be lower than the existing designs [4,10]. Moreover, although the proposed architecture in its simplest form only processes one character at a time, the architecture can be extended to further enhance the throughput of the circuit. Multiple characters can be scanned and processed in one cycle at the expense of slight increase in area cost.

The proposed architecture has been prototyped and simulated by the Altera Stratix FPGA. Experimental results reveal that the circuit attains the throughput up to 6.75 Gbits/sec with area cost of 0.70 LE per character. The proposed architecture therefore is an effective solution to high throughput and low area cost NIDS hardware design.

2. Preliminaries

We briefly review the shift-or algorithm for exact string matching in this section. Suppose $P = p_1p_2 \dots p_m$ is a *pattern* to be searched inside a large *text* (or *source*) $T = t_1t_2 \dots t_n$, where $n \gg m$. Every character of P and T belongs to the same alphabet $\Sigma = \{s_1, \dots, s_{|\Sigma|}\}$.

Let R_j be a bit vector containing information about all matches of the prefixes of P that end at j. The vector contains $m+1$ elements $R_j[i], i = 0, \dots, m$, where $R_j[i] = 0$ if the first i characters of the pattern P match exactly the last i characters up to j in the text (i.e., $p_1p_2 \dots p_i = t_{j-i+1}t_{j-i+2} \dots t_j$). The transition from R_j to R_{j+1} is performed by the recurrence:

$$R_{j+1}[i] = \begin{cases} 0, \text{ if } R_j[i-1] = 0 \text{ and } p_i = t_{j+1}, \\ 1, \text{ otherwise,} \end{cases} \quad (1)$$

where the initial conditions for the recurrence are given by $R_0[i] = 1, i = 1, \dots, m$, and $R_j[0] = 0, j = 0, \dots, m$. The recurrence can be implemented by the simple shift and OR operations. To see this fact, we first associate each symbol $s_k \in \Sigma$ a bit vector S_k containing m elements, where the i-th element $S_k[i]$ is given by

$$S_k[i] = \begin{cases} 0, \text{ if } s_k = p_i, \\ 1, \text{ otherwise.} \end{cases} \quad (2)$$

Assume $t_{j+1} = s_c$. Based on Eq. (2), the recurrence shown in Eq. (1) can then be rewritten as

$$R_{j+1}[i] = R_j[i-1] \; OR \; S_c[i], i = 1, \dots, m. \quad (3)$$

We can clearly see now the transition from R_j to R_{j+1} involves to no more than a shift of R_j and an OR operation with S_c, where $t_{j+1} = s_c$. Figure 1 shows an example of the exact string matching based on the shift-or algorithm, where $P = aab$ and $\Sigma = \{a, b, c\}$. The bit vector S_k associated with each $s_k \in \Sigma$, which is determined by Eq. (2), is given in Fig. 1(a). In this example, $T = acaab$. Therefore, $s_c = a, c, a, a$ and b for $j = 1, 2, 3, 4$ and 5, respectively. The S_c associated with s_c for each j can be found from the table shown in Fig. 1(a). Given S_c and R_{j-1}, the R_j can be computed by Eq. (3), as shown in Fig. 1(b). Note that, when $j = 5$, it can be found from Fig. 1(b) that $R_j[3] = 0$. Therefore, one occurrence of P is found when $j = 5$.

3. The architecture

The proposed architecture for SNORT pattern matching is shown in Fig. 2. The architecture contains M modules, where M is the number of SNORT rules for intrusion detection. The incoming source is first broadcasted to all the modules. Each module is responsible for the pattern matching of a single rule. The encoder in the architecture receives the intrusion alarms issued by the modules detecting matched strings, and transfers the alarms to the administrators for proper actions.

s_k		a	b	c
S_k	$i=1$	0	1	1
	$i=2$	0	1	1
	$i=3$	1	0	1

(a)

j		0		1		2		3		4		5	
		R_j	S_c	R_j	S_c	R_j	S_c	R_j	S_c	R_j	S_c	R_j	
i	0	0		0		0		0		0		0	
	1	1	0	0	1	1	0	0	0	0	1	1	
	2	1	0	1	1	1	0	1	0	0	1	1	
	3	1	1	1	1	1	1	1	1	1	0	0	

(b)

Fig. 1. An example of shift-or algorithm with pattern $P = aab$ and text $T = acaab$, (a) The bit vector S_k associated with each symbol $s_k \in \Sigma = \{a, b, c\}$ for the pattern P, (b) The bit vector R_j for the text T, where one occurrence of P is found (encircled).

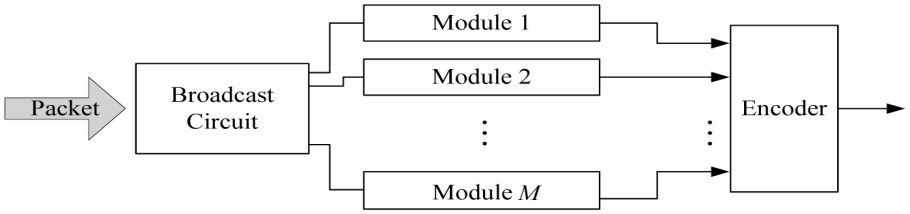


Fig. 2. The basic structure of the proposed circuit, where M is the number of rules implemented by the circuit.

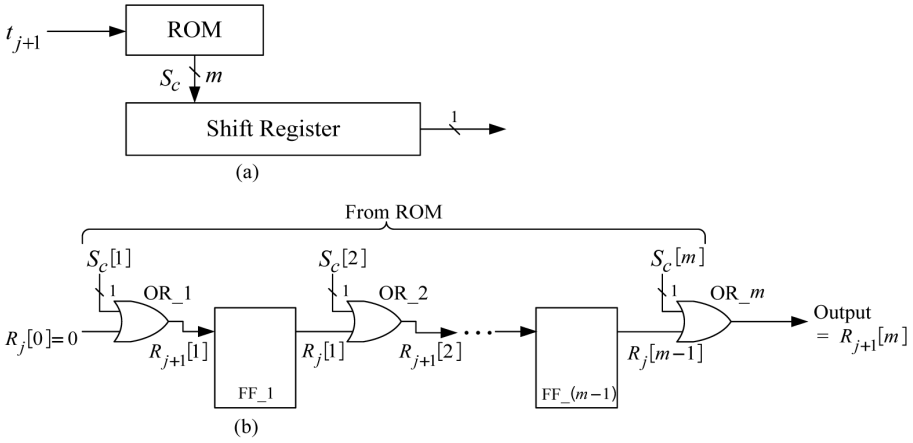


Fig. 3. The basic circuit of each module for exact pattern matching, (a) The block diagram of the circuit, (b) The shift register circuit during clock cycle $j + 1$.

3.1. Basic module circuit

A direct hardware implementation of shift-or algorithm is shown in Fig. 3 [8], where each module contains a ROM and a shift register. There are $|\Sigma|$ entries in the ROM. The k-th entry of the ROM contains the m-bit vector S_k, where m is the size of the pattern associated with the module. The shift register consists of $m-1$ flip-flops (FFs) and m OR gates. Based on the bit vectors $S_k, k = 1, \ldots, |\Sigma|$, provided by the ROM, the objective of the shift register is to perform the shift-or operation shown in Eq. (3).

The module operates by scanning the source string one character at a time. Therefore, *after* the clock cycle j, the circuit completes the string matching process up to t_j. Moreover, the character t_{j+1} is the input character to the module during the clock cycle $(j + 1)$. Assume $t_{j+1} = s_c$. The input character t_{j+1} is first delivered to the ROM for the retrieval of S_c to the OR gates. Each OR gate i has two inputs: one is from the i-th output bit of the ROM (i.e., $S_c[i]$), and the other is from the output of FF $(i-1)$, which contains $R_j[i-1]$ during the clock cycle $j + 1$. From Eq. (3), it follows that the OR gate i produces $R_{j+1}[i]$, which is then used as the input to the FF i. The $R_{j+1}[i]$ therefore will

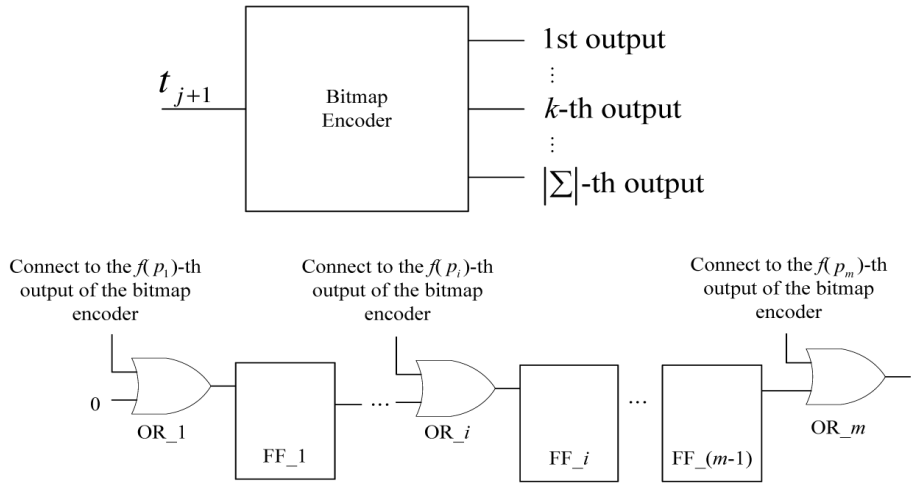


Fig. 4. The shift-or circuit based on a bitmap encoder.

become the output of FF i during the clock $j + 2$ for the subsequent operations.

Note that, during the clock cycle $j + 1$, the m-th OR gate produces $R_{j+1}[m]$, which is identical to 0 when $p_1 p_2 \dots p_i = t_{j-i} t_{j-i+1} \dots t_{j+1}$. In this case, the module will issue an intrusion alarm to the encoder of the NIDS system. Therefore, the output of the OR gate m is the check point of exact string matching with pattern size m.

Although the direct implementation is simple, it has a number of drawbacks. When the ROM is implemented by the LEs, the area cost may be high when the alphabet size $|\Sigma|$ is large. The ROM can be realized by the embedded memory bits. However, the memory bits may operate at slower speed as compared with the LEs. For example, the maximum clock rate of the embedded memory M4K of Altera Stratix 1S40 is only 320 MHz; whereas, the maximum operating frequency of LEs of Altera Stratix 1S40 is 420 MHz. Therefore, the ROM implemented by embedded memory bits may become the bottleneck of the systems's throughput. In addition, the same ROM cannot be shared by different rules. The consumption of embedded memory bits will be high for the circuits containing large number of Snort rules.

3.2. Module circuit based on bitmap encoder

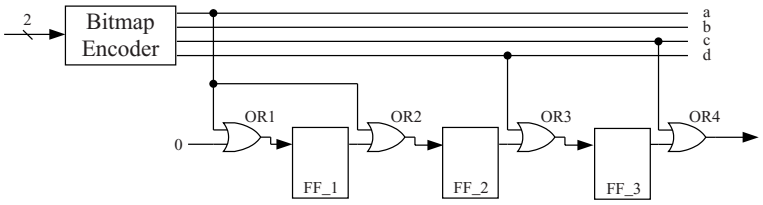
In the proposed circuit, the shift-or circuit is implemented without ROMs. Therefore, no embedded memory bits are required, and higher operating frequencies can be attained. The ROMs are replaced by a simple bitmap encoder, which can be shared by different rules. Consequently, the proposed circuit is well suited for the implementation of systems containing large number of Snort rules.

The bitmap encoder has $|\Sigma|$ bits output. When the symbol s_k is presented at the input, only the k-th bit output of the encoder will be set to zero while the others are set to one. Given a pattern $P = p_1 p_2 \dots p_m$, the simple bitmap encoder can be used to obtain the bit vectors S_k for each symbol $s_k \in |\Sigma|$.

Figure 4 depicts the proposed circuit based on the bitmap encoder. As shown in the figure, the circuit contains only the bitmap encoder and a shift register. The shift register also consists of $m - 1$ FFs and m OR gates. Similar to the ROM-based module, each OR gate i also has two inputs, and the first one is connected to the output of FF $(i - 1)$. However, the second input is connected to the outputs of the bitmap encoder. The connection is dependent on p_i, the i-th character of the pattern P. When $p_i = s_k$, we connect the second input of OR gate i to the k-th output of the bitmap encoder. Define the function $f(p_i) = k$ if $p_i = s_k$. Therefore, as shown in Fig. 4, the second input of the OR gate i is connected to the $f(p_i)$-th output of the bitmap encoder.

Let $\mathcal{I}_k = \{i : f(p_i) = k\}$. Consequently, when s_k is presented at the input of the bitmap encoder, all the OR gates having index $i \in \mathcal{I}_k$ receives 0 from the bitmap encoder, while the others receive 1. Based on Eq. (2), we conclude that each OR gate i obtains $S_k[i]$ from the bitmap encoder when the input symbol is s_k. That is, the bitmap encoder can replace the ROM for hardware shift-or implementation.

Figure 5 shows a simple example of the proposed circuit for the pattern $P = aadc$ and $\Sigma = \{a, b, c, d\}$. In this example, since there are only four symbols, the

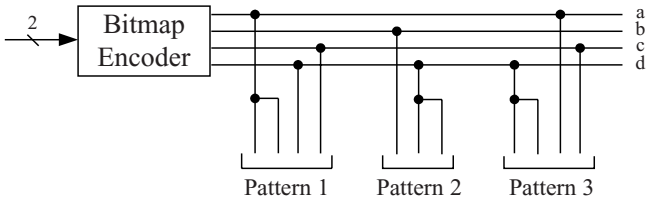


(a)

Input	Pattern 1
a	0011
b	1111
c	1110
d	1101

(b)

Fig. 5. A simple example of the proposed circuit for the pattern $aadc$ and the total symbol a, b, c, d, (a)The architecture (b)Table of the pattern.



(a)

Input	Pattern 1	Pattern 2	Pattern 3
a	0011	111	1101
b	1111	011	1111
c	1110	110	1110
d	1101	101	0011

(b)

Fig. 6. An example of three patterns ($aadc$, bdd and $ddac$) share the same bitmap encoder, (a)The architecture (b)Table of three patterns.

bitmap encoder has four outputs. We set $a = s_1, b = s_2, c = s_3$ and $d = s_4$. Because $p_1 = p_2 = a$, we have $f(p_1) = f(p_2) = 1$. Both the OR1 and OR2 are connected to the first output of the bitmap encoder. Moreover, based on the facts that $p_3 = d$ and $p_4 = c$, we connect the OR3 and OR4 to the 4th and 3rd outputs of the bitmap encoder, respectively. The shift register therefore can receive bit vectors 0011 (i.e., S_1), 1111 (i.e., S_2), 1110 (i.e., S_3) and 1101 (i.e., S_4) when symbols a, b, c and d are the input symbols, respectively.

One advantage of using the bitmap encoder is that the encoder is independent of the pattern P. Only the connection between the encoder and shift register is dependent on P. Consequently, different Snort rules can share the same bitmap encoder. An example of three patterns ($aadc$, bdd and $ddac$) sharing the same bitmap encoder is shown in Fig. 6, where we set $\Sigma = \{a, b, c, d\}$.

To implement the bitmap encoder, we first note that each ASCII character in a Snort rule contains 8 bits. Therefore, $|\Sigma| = 256$ and the bitmap encoder contains 256 outputs for pattern matching. The area complexity of bitmap encoder can be reduced by observing the fact that some symbols s_k in the alphabet Σ may not appear in the pattern P. These symbols then can share the same output in the bitmap encoder for complexity reduction. One simple way to accomplish this is to augment a new symbol s_0 in the alphabet Σ. All the symbols s_k having $S_k = \mathbf{1}$ are then mapped to s_0 by a symbol encoder as shown in Fig. 7. These symbols then shared the same output associated with s_0 in the bitmap encoder.

Although using the symbol encoder can reduce the area cost of bitmap encoder, when rules have different sets of unused symbols, the sharing of symbol encoder and bitmap encoders by different rules may be difficult. One way to solve this problem is to first divide the Snort rules into several groups, where the rules in each group use the same set of symbols. All the rules in

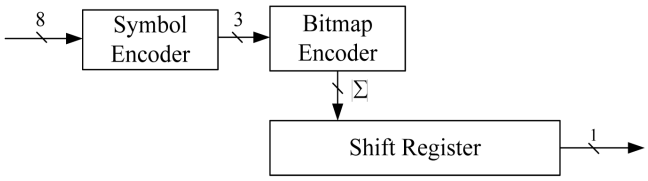


Fig. 7. The increase of a symbol encoder for reducing the bitmap encoder size. In this example, each input character is assumed to be an ASCII code (8 bits). We also assume the Snort rule uses only 7 symbols in the alphabet. The output of the symbol encoder is 3 bits.

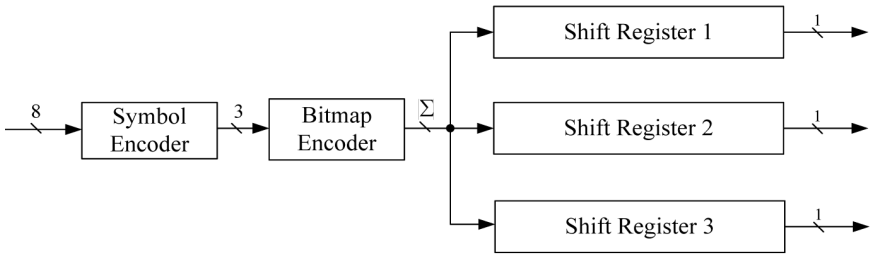


Fig. 8. The sharing of the same symbol encoder and bitmap encoder by three different Snort rules. Each character is also assumed to be an ASCII. All the Snort rules use the same alphabet comprised of 7 symbols.

the same group can then share the same symbol and bitmap encoders, as shown in Fig. 8. The overhead for the realization of encoders then can be reduced.

3.3. High throughput module circuit

The basic module circuit shown in Fig. 3 only process one character per cycle. The throughput of the NIDS system can be improved further by processing q characters at a time. This can be accomplished by grouping q consecutive characters in the source into a single symbol. Without loss of generality, we consider $q = 2$. Let $\Omega = \{x_1, \ldots, x_{|\Omega|}\}$ be the alphabet for the new symbols, where $x_i = (y_1, y_2)$, and $y_1, y_2 \in \Sigma$.

Based on Ω, a pattern P can be rewritten as $P = u_1 u_2 \ldots u_{\lceil m/2 \rceil}$, where $u_i = (p_{2i-1}, p_{2i})$. Note that $u_{\lceil m/2 \rceil} = (p_{m-1}, p_m)$ when m is even. However, when m is odd, $u_{\lceil m/2 \rceil} = (p_m, \phi)$, where ϕ denotes "don't care," and can be any character in Σ. We can then associate a bit vector X_k containing $\lceil m/2 \rceil$ elements for each symbol $x_k \in \Omega$, where the i-th element of X_k is given by

$$X_k[i] = \begin{cases} 0, \text{ if } x_k = u_i, \\ 1, \text{ otherwise.} \end{cases} \quad (4)$$

A bitmap encoder providing $X_1, \ldots, X_{|\Omega|}$ can then be constructed for shift-or operations. In this case, the bitmap encoder contain $|\Omega| = |\Sigma|^2$ outputs, where each output i will be set to 0 when the symbol x_i is presented at the input. It is therefore necessary to employ a larger bitmap encoder for a module with higher throughput. A symbol encoder similar to that shown in Fig. 7 can be employed to reduce the complexity of bitmap encoder. In this case we augment a new symbol x_0 (with $X_0 = \mathbf{1}$) in the alphabet Ω. All the symbols x_k having $X_k = \mathbf{1}$ are then mapped to x_0 by the symbol encoder.

Note that the string matching operations ending at j over the alphabet Ω is equivalent to the operations ending at either $2j$ or $2j + 1$ (but not both) over the alphabet Σ. It is necessary to perform the matching process ending at every location of the source over the alphabet Σ. Therefore, we employ two shift registers in the module as shown in Fig. 9, where one is for even locations, and the other is for odd locations.

Moreover, since the pattern P in this case contains $\lceil m/2 \rceil$ symbols, the shift registers with $\lceil m/2 \rceil - 1$ FFs and $\lceil m/2 \rceil$ OR gates are sufficient for the operations. Therefore, the total number of FFs in the high throughput circuit is $2\lceil m/2 \rceil - 2$, which is less than that in the basic circuit presented in the previous subsection.

To perform the string matching operations ending at the *even* locations of the source over Σ, we convert the source T to the sequence $T_e = e_1 e_2 \ldots$ over alphabet Ω, where $e_j = (t_{2j-1}, t_{2j})$. During the clock cycle $j + 1$, symbol e_{j+1} is fetched to the symbol encoder. This is equivalent to the scanning of two characters t_{2j+1} and t_{2j+2} simultaneously for shift-or operations.

The shift-or operations at the *odd* locations of the source can be performed in the similar manner, except that the source T is extracted as $T_o = o_1 o_2 \ldots$, where $o_j = (t_{2j}, t_{2j+1})$. During the clock cycle $j + 1$, we

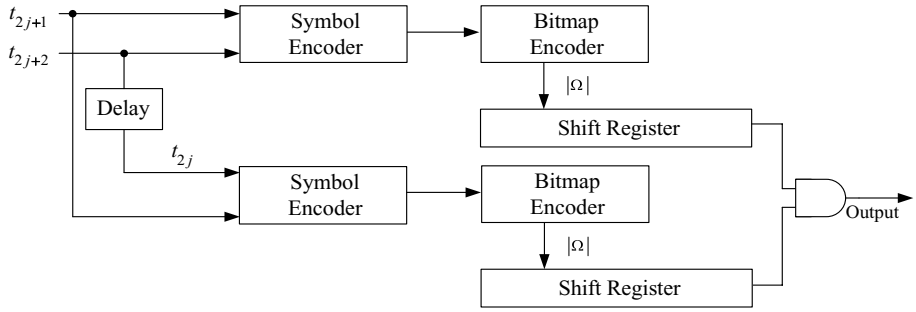


Fig. 9. The structure of a high throughput module circuit processing two characters at a time ($q = 2$) with the bitmap encoder.

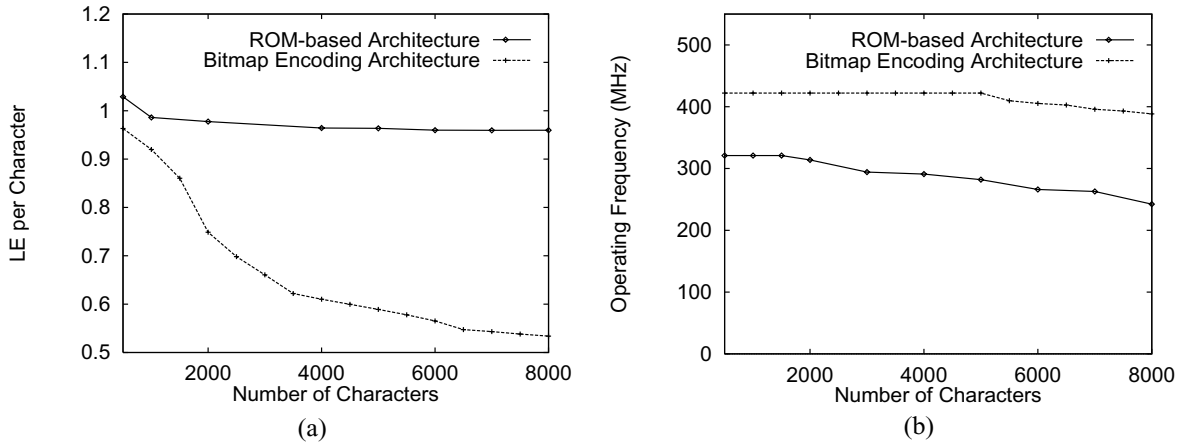


Fig. 10. The performance of the ROM-based and bitmap encoding circuit with $q = 1$ for various rule sets sizes ranging from 500 characters to 8000 characters (a) LE per character (b) Operating Frequency.

scan the symbol o_j. From Fig. 9, we observe that o_j can be obtained from e_j and e_{j+1} via delaying and broadcasting operations. Therefore, the shift-or operations at even and odd locations share the same input as shown in the figure.

4. Experimental results and comparisons

Figure 10 shows the average number of LEs per character and operating frequency of the proposed circuit with $q = 1$ for various rule sets with sizes ranging from 200 characters to 8000 characters. In this experiment, both the symbol encoder and bitmap encoder are shared by different rules for reducing the area cost for the FPGA implementation. In addition to the proposed circuit, the ROM-based shift-or circuit [8] is considered in the figure for comparison purpose. In the ROM-based circuit, the symbol encoder and ROM are also shared by different rules. We use the Altera Quartus II as the tool for circuit synthesization. The target FPGA device is Stratix EP1S40.

From Fig. 10, it can be observed that the operating frequency of the proposed circuit is stable over a wide range of rule set sizes. Moreover, the average number of LEs per characters decreases as the size of rule set increases. This is because the area overhead for implementing the symbol encoder reduces as the number of rules sharing the encoder increases. In addition, similar rules may share the same portion of a shift register.

Given the same size of rule set, it can be observed from Fig. 10 that the proposed architecture has higher operating frequency as compared with the ROM-based architecture. The proposed architecture has faster operating frequency because it contains no embedded memory. In many FPGA devices such as Stratix EP1S40, the embedded memory operates at slower clock rate as compared with the usual LEs. In addition, circuits containing only LEs may provide higher flexibility for the place and route optimization as compared with the circuits consisting of both the LEs and embedded memory blocks. Consequently, the proposed architecture has lower area complexity as shown in Fig. 10.

Table 1
Comparisons of various string matching FPGA designs, where the number of characters available for pattern matching is 1568 characters

Design	Device	Throughput (Gb/s)	Memory Bits	LEs /char	Operating Frequency (MHz)
Proposed architecture ($q = 1$)	Altera Stratix EP1S40	3.38	0	0.7	422.12
Proposed architecture ($q = 2$)	Altera Stratix EP1S40	6.75	0	0.7	422.12
ROM-based shift-OR architecture ($q = 2$) [8]	Altera Stratix EP1S40	5.14	40768	1.09	321.03

Table 2
Comparisons of various string matching FPGA designs.

Design	Device	Throughput (Gb/s)	No. characters	LEs /char
Proposed architecture ($q = 1$)	Altera Stratix EP1S40	3.11	8000	0.53
Proposed architecture ($q = 2$)	Altera Stratix EP1S40	6.75	1568	0.7
Clark-Schimmel [2]	Xilinx Virtex2-8000	2.2	7996	1.88
Cho et al. [3]	Xilinx Spartan3-2000	3.2	6805	0.9
Gokhale et al. [4]	Xilinx VirtexE-1000	2.2	640	15.2
Hutchings et al. [5]	Xilinx Vertix-1000	0.248	8003	2.57
Moscola et al. [6]	Xilinx VirtexE-2000	1.18	420	19.4
Singaraju et al. [9]	Xilinx Virtex2VP30-7	6.41	1021	2.2
Sourdis-Pnevmatikatos [10]	Xilinx Spartan33-5000	4.91	18000	3.69

Table 1 shows the throughput, the average number of LEs per character, total number of memory bits and operating frequency of the various shift-or circuits with $q = 1$ or 2. All the circuits have the same rule set size 1568 characters. In the table, the throughput indicates the maximum number of bits per second the circuit can process. As shown in Table 1, because the circuit with $q = 2$ processes two characters for each clock cycle, it has higher throughput than that of the circuit with $q = 1$, which processes one character per cycle only.

From Table 1, it can also be observed that, when $q = 2$ and rule size 1568 characters, the ROM-based architecture requires 40768 memory bits for attaining the throughput 5.14 Gbits/sec and area complexity 1.09 LEs per character. Based on the same q and rule set size, the proposed architecture consumes no memory bits for achieving higher throughput 6.75 Gbits/sec and lower area cost 0.70 LE per character. Therefore, we conclude that the ROM-based architecture requires large number of memory bits for the hardware implementation of shift-or algorithm. By contrast, the proposed architecture attains higher throughput and lower average number of LEs per character without the consumption of memory bits.

Table 2 compares the FPGA implementations of the proposed architecture with those of the existing related works. Note that the exact comparisons of the proposed circuits with the related work may be difficult because they are realized by different FPGA devices. However, it can still be observed from the table that our circuits have effective throughput-area performance as compared with existing work. This is because our design is based on the simple shift-or algorithm. The simplicity of circuit allows the string matching operations to be performed at high clock rate with small hardware area. In fact, with $q = 1$, the proposed architecture requires lowest average LEs per character. When $q = 2$, the proposed architecture also has superior throughput over the existing architectures. These facts show the effectiveness of the proposed architecture.

5. Conclusion

A novel FPGA implementation of NIDS systems based on shift-or algorithm is presented in this paper. The proposed algorithm in the basic form process one character at a time, and contain only a ROM and a simple shift register for each pattern matching. The throughput can be further enhanced by replacing the ROM with a simple bitmap encoder, and by processing multiple characters in parallel. Both the one-character and two-character at a time of the proposed algorithm are implemented in our experiments. Comparisons with existing work reveal that our design is one of the cost-effective solutions to the FPGA implementations of the NIDS systems.

References

[1] R. Baeza-Tates and G.H. Gonnet, A new approach to text searching, *Communications of the ACM* **35** (1992), 74–82.

[2] C. Clark and D. Schimmel, Scalable multi-pattern matching on high speed networks, *Proceedings of the IEEE Symposium on Field- Programmable Custom Computing Machines* (2004), 249–257.

[3] Y.H. Cho and W.H. Mangione-Smith, Deep packet filter with dedicated logic and read only memories, *Proceedings of the IEEE Symposium on Field- Programmable Custom Computing Machines* (2004), 125–134.

[4] M. Gokhale, D. Dubois, A. Dubois, M. Boorman, S. Poole and V. Hogsett, Granidt: towards gigabit rate network intrusion detection technology, *Proceedings of the International Conference on Field Programmable Logic and Application* (2002), 404–413.

[5] B.L. Hutchings, R. Franklin and D. Carver, Assisting network intrusion detection with reconfigurable hardware, *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines* (2002), 111–120.

[6] J. Moscola, J.W. Lockwood, R.P. Loui and M. Pachos, Implementation of a Content-Scanning Module for an Internet Firewall, *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines* (2003), 31–38.

[7] T. Ramirez and C.D. Lo, Rule Set Decomposition for Hardware Network Intrusion Detection, *in the 2004 International Computer Symposium* (*ICS 2004*), Dec. 15–17, 2004, Taipei, Taiwan, 2004.

[8] H.C. Roan, C.M. Ou, W.J. Hwang and C.T.D. Lo, Efficient Logic Circuit for Network Intrusion Detection, *Lecture Notes in Computer Science* **4096** (2006), 776–784.

[9] J. Singaraju, L. Bu and J.A. Chandy, A signature match processor architecture for network intrusion detection, *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines* (2005), 235–242.

[10] I. Sourdis and D.N. Pnevmatikatos, Pre-decoded CAMs for efficient and high-speed NIDS pattern matching, *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines* (2004), 258–267.

[11] SNORT official web site. http://www.snort.org.