

## Short Paper

---

### Network Intrusion Detection Based on Shift-OR Circuit<sup>\*</sup>

HUANG-CHUN ROAN, WEN-JYI HWANG, WEI-JHIH HUANG AND CHIA-TIEN DAN LO<sup>†</sup>

*Department of Computer Science and Information Engineering  
National Taiwan Normal University  
Taipei, 117 Taiwan*

*<sup>†</sup>Department of Computer Science  
University of Texas at San Antonio  
San Antonio, TX 78249, U.S.A.*

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 key feature of the signature match co-processor is an architecture based on the shift-or algorithm, which employs simple shift registers, or-gates, and ROMs where patterns are stored. 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 [8] 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 SNORT system running on general purpose processors may only achieve up to 60 Mbps [5] 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 [2, 3, 5-7]. FPGA-based reconfigurable hardware can be programmed almost like software, maintaining the most attractive advantage of flexibility with less cost than traditional ASIC hardware implementations. Moreover, the FPGA

---

Received July 4, 2006; revised September 26, 2006; accepted October 25, 2006.

Communicated by Tzong-Chen Wu.

<sup>\*</sup>This paper was presented in part at the IEEE International Conference on Field Programmable Logic and Applications (FPL 2006), Madrid Spain, August 2006. This project was partially supported by the Center for Infrastructure Assurance and Security at UTSA and US Air Force under grant #26-0200-62.

hardware implementation can exploit parallelism for string matching so that the throughput of NIDSs can be increased.

One popular way for FPGA implementation is based on regular expressions [3, 4], which results in designs with low area cost and moderate throughput acceleration. In this approach, a regular expression is generated for every pattern. Each regular expression is then implemented by a nondeterministic finite automata (NFA) or deterministic finite automata (DFA). 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) [2, 7]. 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.

The objective of this paper 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.

On the other hand, the hardware implementation of shift-or algorithm imposes no limitation on the pattern size. In our architecture, each SNORT pattern is only associated with a ROM and a shift register for pattern comparison, which are designed in accordance with the pattern size. Because of its simplicity, the architecture may operate at a higher clock rate as compared with other implementations. In addition, the number of logic elements (LEs) for the circuit implementation is reduced significantly when the ROM is realized by the embedded RAM blocks of the FPGA. The area cost therefore may be lower than the existing designs [2, 7]. 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 5.14 Gbits/sec with area cost of 1.09 LE per character. The proposed architecture therefore is an effective solution to high throughput and low area cost NIDS hardware design.

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

(a)

		$j$											
		0		1		2		3		4		5	
		$R_j$	$S_c$	$R_j$	$S_c$	$R_j$	$S_c$	$R_j$	$S_c$	$R_j$	$S_c$	$R_j$	$S_c$
$S_k$	$i$	0	0	0	0	0	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).

**2. PRELIMINARIES**

This section briefly describes the shift-or algorithm for exact string matching. Suppose we are searching for a *pattern*  $P = p_1p_2 \dots p_m$  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:

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

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

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

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

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

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  $t_{j+1} = s_c$ . Fig. 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  $j = 5$ .

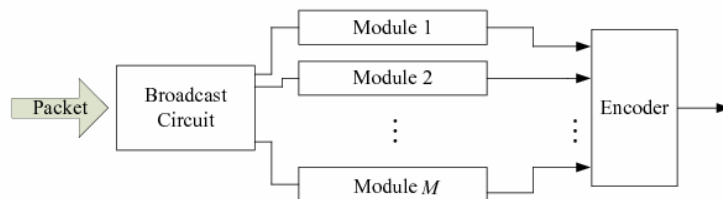


Fig. 2. The basic structure of the proposed circuit, where  $M$  is the number of rules implemented by the circuit.

### 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.

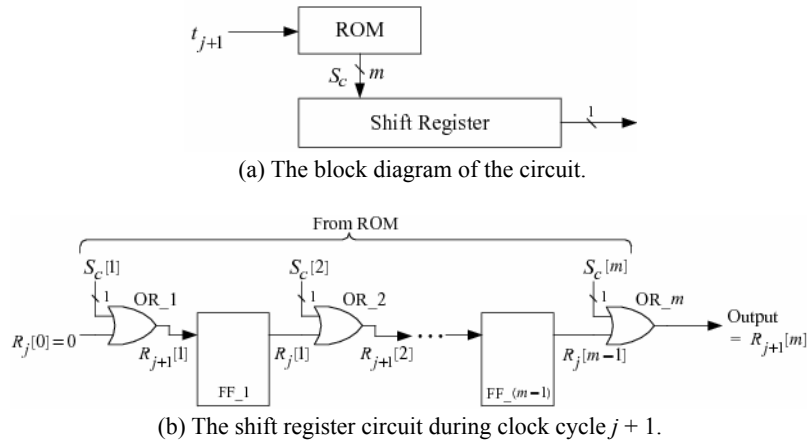


Fig. 3. The basic circuit of each module for exact pattern matching.

#### 3.1 Basic Module Circuit

Each module uses the shift-or algorithm for exact string matching in hardware. As shown in Fig. 3, 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  $S_k$ ,  $k = 1, \dots, |\Sigma|$ , 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 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  $p_1 p_2 \dots p_i = t_j t_{j-i+1} \dots t_{j+1}$ . 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  $m$ .

For the FPGA devices with embedded memories, the ROM may be implemented solely by the memory bits. Hence, the LEs are required only for the implementation of the shift register. The circuit therefore may have low area cost (in terms of the number of LEs) for the FPGA implementation of SNORT rules.

To implement the ROM, we first note that each ASCII character in a SNORT rule contains 8 bits. Therefore,  $|\Sigma| = 256$  and the ROM contains 256 entries for pattern matching. The ROM size can be reduced by observing the fact that some symbols  $s_k$  in the alphabet  $\Sigma$  may not appear in the pattern  $P$ . Accordingly, they have the same bit vectors  $S_k = 1$ . These symbols then can share the same entry in the ROM for storage size reduction. One simple way to accomplish this is to augment a new symbol  $s_0$  (with  $S_0 = 1$ ) in the alphabet  $\Sigma$ . All the symbols  $s_k$  having  $S_k = 1$  are then mapped to  $s_0$  by a symbol encoder as shown in Fig. 4. These symbols then shared the same entry associated with  $s_0$  in the ROM.

Since the LEs are required for the implementation of symbol encoders, the area cost may be high if each module has its own symbol encoder. We can lower the area cost by first dividing the SNORT rules into several groups, where the rules in each group use the same set of symbols. Therefore, all the rules in the same group can share the same symbol encoder, as shown in Fig. 5. The overhead for the realization of symbol encoders then can be reduced.

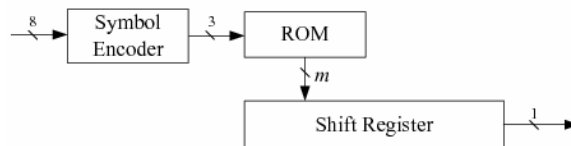


Fig. 4. The augment of a symbol encoder for reducing the ROM size. In this example, each input character is assumed to be an ASCII code (8 bits). We also assume the SNORT rule uses only 7 symbols in the alphabet. The output of the symbol encoder therefore is 3 bits.

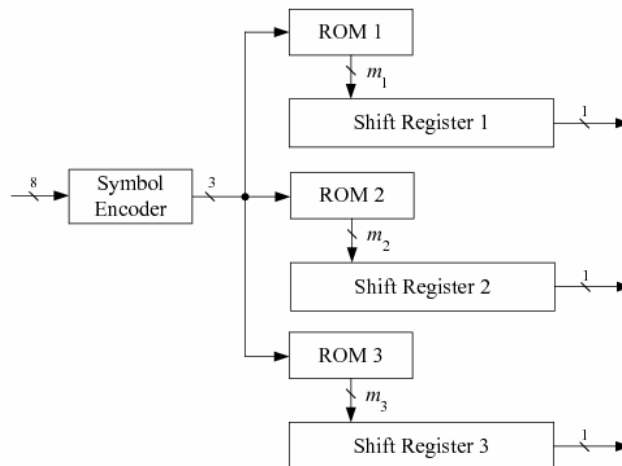


Fig. 5. The sharing of the same symbol encoder and bitmap encoder by three different Snort rules. Each character is also assumed to be an ASCII. All the Snort rules use the same alphabet comprised of 7 symbols.

To partition the SNORT rules, we first note that the number of rules is finite. Therefore, a simple supervised rule partitioning method is adopted here. In the algorithm, first we obtain the distribution of each symbol among different rules. Based on the distributions, the number of rule groups and the set of symbols associated with each group are then determined by inspection.

### 3.2 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  $\Omega = \{x_1, \dots, x_{|\Omega|}\}$  be the alphabet for the new symbols, where  $x_i = (y_1, y_2)$ , and  $y_1, y_2 \in \Sigma$ .

Based on  $\Omega$ , a pattern  $P$  can be rewritten as  $P = u_1 u_2 \dots u_{\lceil m/2 \rceil}$ , 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  $\phi$  denotes “don’t care” and can be any character in  $\Sigma$ . 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 ROM containing  $X_1, \dots, X_{|\Omega|}$  can then be constructed for shift-or operations. In this case, the ROM contain  $|\Omega| = |\Sigma|^2$  entries, where each entry has  $\lceil m/2 \rceil$  bits. It is therefore necessary to employ a larger ROM for a module with higher throughput. A symbol encoder similar to that shown in Fig. 4 can be employed to reduce the ROM size. In this case we augment a new symbol  $x_0$  (with  $X_0 = 1$ ) in the alphabet  $\Omega$ . All the symbols  $x_k$  having  $X_k = 1$  are then mapped to  $x_0$  by the symbol encoder.

Note that the string matching operations ending at  $j$  over the alphabet  $\Omega$  is equivalent to the operations ending at either  $2j$  or  $2j + 1$  (but not both) over the alphabet  $\Sigma$ . It is necessary to perform the matching process ending at every location of the source over the alphabet  $\Sigma$ . Therefore, we employ two shift registers in the module as shown in Fig. 6, where one is for even locations, and the other is for odd locations. Moreover, since each entry of the ROM contains only  $\lceil m/2 \rceil$  bits, 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  $\Sigma$ , we convert the source  $T$  to the sequence  $T_e = e_1 e_2 \dots$  over alphabet  $\Omega$ , where  $e_j = (t_{2j-1}, t_{2j})$ . During the clock cycle  $j + 1$ , symbol  $e_{j+1}$  is fetched to the ROM. 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  $T_o = o_1 o_2 \dots$ , where  $o_j = (t_{2j}, t_{2j+1})$ . During the clock cycle  $j + 1$ , we scan the symbol  $o_j$ . From Fig. 6, 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.

It can be observed from Fig. 6 that two identical ROMs are required for concurrent reads for each rule. The storage overhead may be reduced further by the employment of a dual-port ROM allowing the same memory block to be shared by two concurrent reads, as shown in Fig. 7. An example of the embedded memory blocks supporting the realization of dual-port ROM is the M4K blocks of Altera Stratix FPGA devices, where a true dual-port mode supporting any combination of two-port operations (*i.e.*, two reads, two writes, or one read and one write) is provided. The utilization of these embedded memory blocks is very helpful for the implementation of the proposed circuits achieving both high throughput and low area cost.

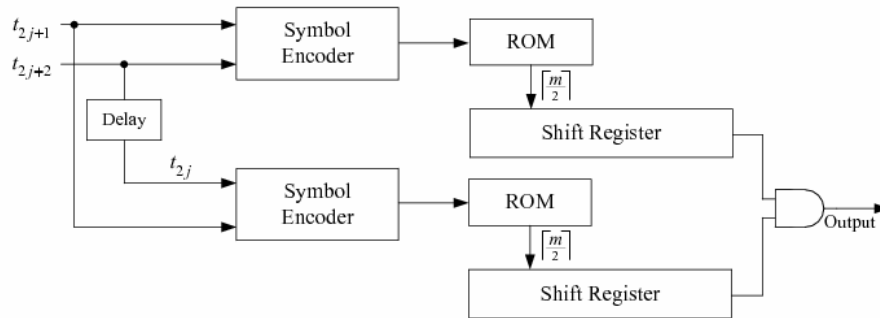


Fig. 6. The structure of a high throughput module circuit processing two characters at a time ( $q = 2$ ).

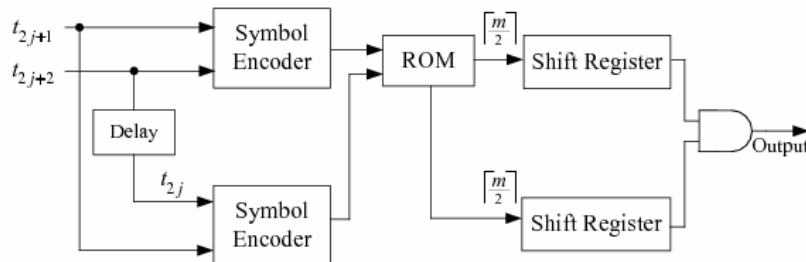


Fig. 7. The structure of a high throughput module circuit processing two characters at a time ( $q = 2$ ) with a shared dual-port ROM.

#### 4. EXPERIMENTAL RESULTS AND COMPARISONS

This section presents experimental results of the proposed architecture for NIDS. All the rules considered here are used for SNORT 2.2. Fig. 8 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 500 characters to 8000 characters. In this experiment, the symbol encoder is used to reduce the storage size of the ROM. In addition, different rules will share the same symbol encoder for reducing the area cost for the FPGA implementation. We use the Altera Quartus II as the tool for circuit synthesization. The target FPGA device is Stratix EP1S40.

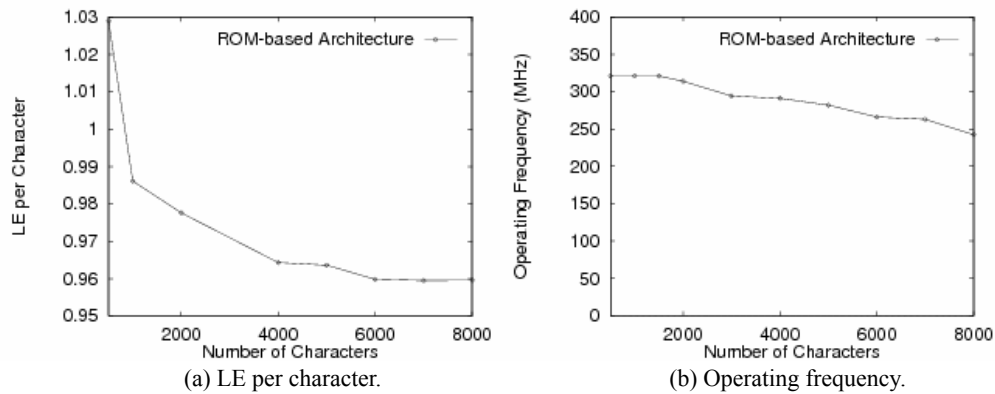


Fig. 8. The performance of the proposed circuit with  $q = 1$  for various rule sets with sizes ranging from 500 characters to 8000 characters.

**Table 1. Comparisons of the proposed architecture with  $q = 2$  for various configurations.**

Configurations			Throughput (Gb/s)	LEs/char	Memory bits	Operating Frequency (MHz)
Symbol Encoder		ROM Sharing				
Utilization	Sharing					
No	No	No	Not available	Not available	102,760,448	Not available
Yes	No	No	3.56	1.74	39,718	222.77
Yes	Yes	No	5.14	1.09	40,768	321.03
Yes	Yes	Yes	4.65	1.08	20,826	290.87

From Fig. 8, 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 character 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 particular, when the rule set size is 8000 characters, the average number of characters becomes only 0.95 LE/char.

Table 1 compares the throughput, the average number of LEs per character, total number of memory bits and operating frequency of the proposed circuits for various configurations. Only the circuits processing two characters at a time (*i.e.*,  $q = 2$ ) are considered in the table. The rule set size is 1568 characters. In the table, the throughput indicates the maximum number of bits per second the circuit can process.

Because the alphabet size is  $2^{16}$  for  $q = 2$ , when the symbol encoder is not utilized, the ROMs for each rule has  $2^{16}$  entries, resulting in total amount of 102.76M bits for the rule set size of 1,568 characters. Due to large amount of embedded memory bits required for pattern storing, it is difficult to implement the circuit using the existing FPGA devices. As shown in Table 1, the employment of symbol encoder significantly reduce the number of memory bits for ROM implementation (from 102.76M bits to 40.76K bits). Nevertheless, without the sharing of symbol encoder by different rules, the number of LEs consumed by the circuit is 1.74 LEs/char. When the symbol encoder is shared, the area cost is then reduced to 1.09 LEs/char. Moreover, the circuit with symbol encoder sharing

achieves clock rate up to 321.03MHz, which is significantly higher than that of the circuit without symbol encoder sharing.

When the ROM is also shared by string matching operations ending at even and odd locations for each rule, as shown in Fig. 7, the number of memory bits can be reduced further by half (from 40,768 bits to 20,826 bits). Nevertheless, for the Stratix FPGA devices, the ROM sharing is implemented by true dual-port ROMs, which are supported only by M4K embedded memory blocks. On the contrary, the implementation of single-port ROM can be realized by embedded memory blocks with faster speed, such as M512. Therefore, the proposed circuit with ROM sharing operates at slightly slower clock rate as compared with its counterpart without ROM sharing, where the ROMs are implemented by M512.

Table 2 compares the FPGA implementations of the proposed architecture with those of the existing related works. The proposed circuits considered here are implemented with symbol encoder sharing. When  $q = 2$ , the circuits with and without ROM sharing are included. As shown in Table 2, 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. On the other hand, it can also be observed from Table 2 that the circuit with  $q = 2$  has slighter higher number of LEs per character. This is because the circuit has more complex address encoder for reducing the storage size in ROM.

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 particular, when  $q = 2$  without ROM sharing, our circuit attains the throughput of 5.14 Gbits/sec while requiring only the area cost of 1.09 LEs per character. These facts demonstrate the effectiveness of our design.

**Table 2. Comparisons of various string matching FPGA designs.**

Design	Device	Throughput (Gb/s)	No. characters	Logic cells /char
Proposed architecture ( $q = 1$ )	Altera Stratix EP1S40	2.25	5004	0.96
Proposed architecture ( $q = 2$ ) with ROM sharing	Altera Stratix EP1S40	5.14	1568	1.09
Proposed architecture ( $q = 2$ ) with ROM sharing	Altera Stratix EP1S40	4.65	1568	1.08
Gokhale <i>et al.</i> [2]	Xilinx VirtexE-1000	2.2	640	15.2
Hutchings <i>et al.</i> [3]	Xilinx Vertix-1000	0.248	8003	2.57
Moscola <i>et al.</i> [4]	Xilinx VirtexE-2000	1.18	420	19.4
Singaraju <i>et al.</i> [6]	Xilinx Virtex2VP30-7	6.41	1021	2.2
Sourdis-Pnevmatikatos [7]	Xilinx Spartan33-5000	4.91	1800	3.69

## 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 processing multiple characters in parallel. Both the basic form and two-character at a time of the proposed algorithm are implemented in our experiments. Comparisons with existing work reveal that our design is a cost-effective solution to the FPGA implementation of packet payload string matching for NIDS systems.

Note that circuits performing only packet payload string matching may issue false alarms for some SNORT rules. Future work therefore is desired to incorporate both header and payload matching for reducing the number of false alarms. It would also be interesting to realize stateful packet inspection for stateful SNORT rules.

## REFERENCES

1. R. Baeza-Tates and G. H. Gonnet, "A new approach to text searching," *Communications of the ACM*, Vol. 35, 1992, pp. 74-82.
2. M. Gokhale, D. Dubois, A. Dubois, M. Boorman, S. Poole, and V. Hogsett, "Granidt: towards gigabit rate network intrusion detection technology," in *Proceedings of the International Conference on Field Programmable Logic and Application*, 2002, pp. 404-413.
3. B. L. Hutchings, R. Franklin, and D. Carver, "Assisting network intrusion detection with reconfigurable hardware," in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, 2002, pp. 111-120.
4. J. Moscola, J. W. Lockwood, R. P. Loui, and M. Pachos, "Implementation of a content-scanning module for an internet firewall," in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, 2003, pp. 31-38.
5. T. Ramirez and C. D. Lo, "Rule set decomposition for hardware network intrusion detection," in *Proceedings of the International Computer Symposium*, 2004.
6. J. Singaraju, L. Bu, and J. A. Chandy, "A signature match processor architecture for network intrusion detection," in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, 2005, pp. 235-242.
7. I. Sourdis and D. N. Pnevmatikatos, "Pre-decoded CAMs for efficient and high-speed NIDS pattern matching," in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, 2004, pp. 258-267.
8. SNORT official, <http://www.snort.org>.

**Huang-Chun Roan (阮煥鈞)** received the B.S. degree in Computer Science and Information Engineering with a Business Administration minor from Tamkang University, Taipei, Taiwan, in 2004. He received the M.S. degree in Computer Science and Information Engineering at National Taiwan Normal University. His research interest is focused on reconfigurable hardware design.

**Wen-Jyi Hwang (黃文吉)** received his diploma in Electronics Engineering from National Taipei Institute of Technology, Taiwan, in 1987, and M.S.E.C.E. and Ph.D. degrees from the University of Massachusetts at Amherst in 1990 and 1993, respectively. From September 1993 until January 2003, he was with the Department of Electrical Engineering, Chung Yuan Christian University, Taiwan. In February 2003, he joined the Department of Computer Science and Information Engineering, National Taiwan Normal University, where he is now a full Professor and Chairman of the department. Dr. Hwang is the recipient of the 2000 Outstanding Research Professor Award from Chung Yuan Christian University, 2002 Outstanding Young Researcher Award from the Asia-Pacific Board of the IEEE Communication Society, and 2002 Outstanding Young Electrical Engineer Award from the Chinese Institute of the Electrical Engineering. His research interests include multimedia communications, VLSI and SoC design, video coding standards, and medical signal processing.

**Wei-Jhih Huang (黃威智)** received his B.S. degree in Computer Science from Aletheia University, Taiwan, in 2005. He is currently pursuing M.S. degree in the Department of Computer Science and Information Engineering, National Taiwan Normal University. His research interests include FPGA and VLSI implementations.

**Chia-Tien Dan Lo (羅佳田)** received a B.S. degree in Applied Mathematics from National ChungHsing University, Taiwan, in 1990, a master degree of Computer Science in Electrical Engineering from National Taiwan University, Taiwan, in 1992, and a Ph.D. degree in Computer Science from Illinois Institute of Technology in 2001. He has been an instructor of Computer Science since 1999. Courses taught include software engineering, software validation and quality assurance, assembly language and computer architecture, processor design using VHDL, operating systems, Java programming, queuing theory and Unix system programming. He is currently with the Department of Computer Science, University of Texas at San Antonio as an assistant professor. His research interests include reconfigurable computing, embedded system design, computer architecture, concurrent automatic dynamic memory management, multithreaded programming, programming languages, and model checking. Recently, he is working on Java high performance computing, hardware assisted network intrusion detection, and memory compression.