Chapter 3

ROM-based Architecture

The proposed architecture is based on shift-or algorithm. Observed from the previous chapter, shift-or algorithm requires the table for exact string matching. Moreover, the ROM is exactly a simple table. Therefore, a rom-based architecture is discussed in this chapter. The basic structure of the proposed circuit for Snort pattern matching is illustrated in Figure 3.1. The architecture incorporates M modules, where M indicates the number of Snort rules. At first, the incoming source packet is broadcasted to all the modules. Then, each module deals with the pattern matching of a single rule. The output of each module connects to the encoder. The encoder perceives the intrusion alarms from the module circuit, and then notifies the system administrators for proper actions.

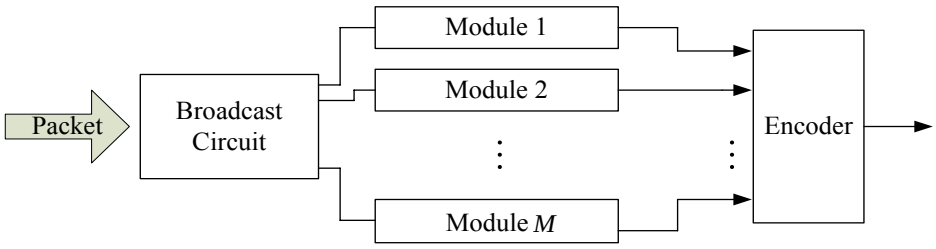

Figure 3.1: The basic structure of the proposed circuit, where M is the number of rules implemented by the circuit.

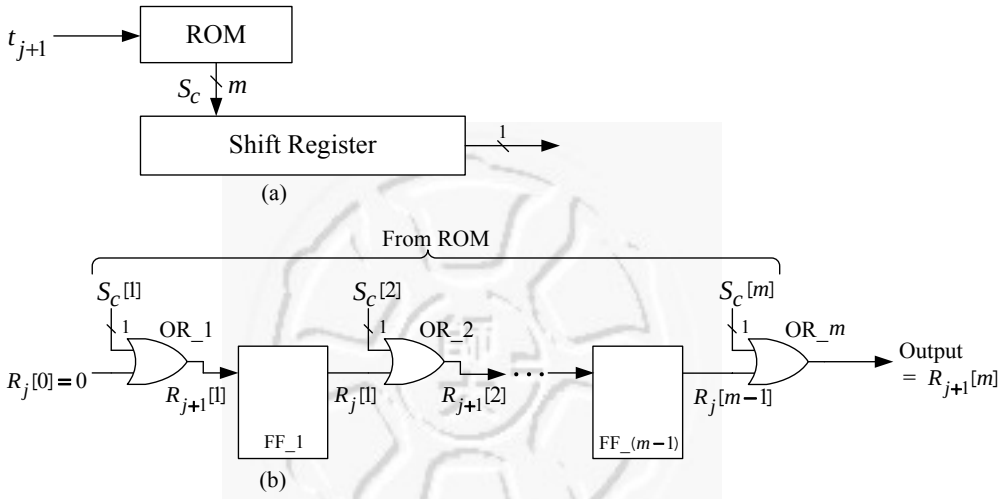

Figure 3.2: The basic circuit of each module for exact pattern matching, (a)The block diagram of the circuit, (b)The shift register circuit during clock cycle $j+1$.

3.1 Basic Module Circuit

Each module adopts the shift-or algorithm for exact string matching. As illustrated in Figure 3.2, each module is comprised of a ROM and a shift register. Within the ROM, there are $|\Sigma|$ entries and the k-th entry occupies bit vector S_k with the pattern size m. Within the shift register, the circuit is composed of $m-1$ flip-flops and m OR gates. The objective of the basic module circuit is to accomplish the shift-or algorithm demostrated in eq.(2.3).

With the basic module, the module can process one input character at a time. Hence, we can complete the string matching procedure over character t_j after the clock cycle j. Moreover, the character t_{j+1} is the input character to the module circuit during the clock cycle $j+1$. As illustrated in Figure 3.2(a), the character t_{j+1} is delivered into the ROM for retrieving the bit vector S_c to the OR gate. As illustrated in Figure 3.2(b), each OR gate includes two inputs. One of them is from the output bit of the ROM, and the other is from the output of previous flip-flop which contains $R_j[i-1]$ during the clock cycle $j+1$. The output of the OR gate is $R_{j+1}[i]$ viewed as the input to the flip-flop i. Then, $R_{j+1}[i]$ will

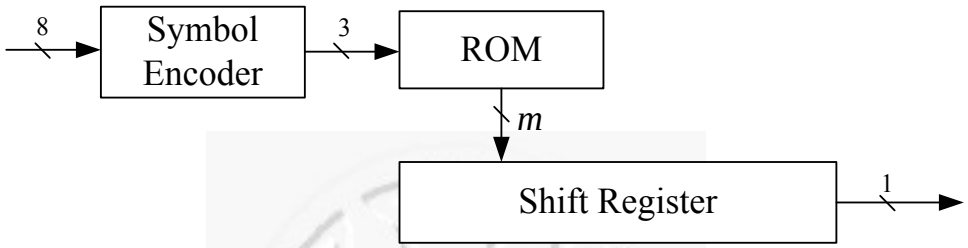

Figure 3.3: The increase of a symbol encoder for reducing the ROM size. In this example, each input character is assumed to be an ASCII code (8 bits). We also assume the Snort rule uses only 7 symbols in the alphabet. The output of the symbol encoder is 3 bits.

become the output of the flip-flop i in the next clock cycle.

Notice that, $R_{j+1}[m]$ generated by the OR gate m is equivalent to zero when the number of m characters of the pattern exactly match the number of m characters up to $j+1$ in the text. In this case, the module will deliver an alarm message to the encoder of NIDS.

Since the FPGA devices own the embedded memory blocks, the ROM could be implemented by the memory bits. Therefore, LEs are required only for the shift register. By the employment of embedded memory blocks, the circuit have low area cost in terms of LEs for the FPGA implementation of Snort rules.

For the ROM implementation, we first note that each ASCII character within a Snort rule contains 8 bits. As a result, the ROM occupies 256 entries for string matching. Actually, we observed from the Snort rule that some symbols in the alphabet are not included in the pattern. Therefore, those unused symbols can share the same entry in the ROM for the purpose to reduce the ROM size. One simple approach to solve the problem is to incoporate a symbol encoder to map those unused symbols into the same entry as illustrated in Figure 3.3.

Since the symbol encoder need LEs for implementation, the area cost may be high if each modules possess its own symbol encoder. We can even lower the area cost by grouping the Snort rules into several partitions. The rules in each

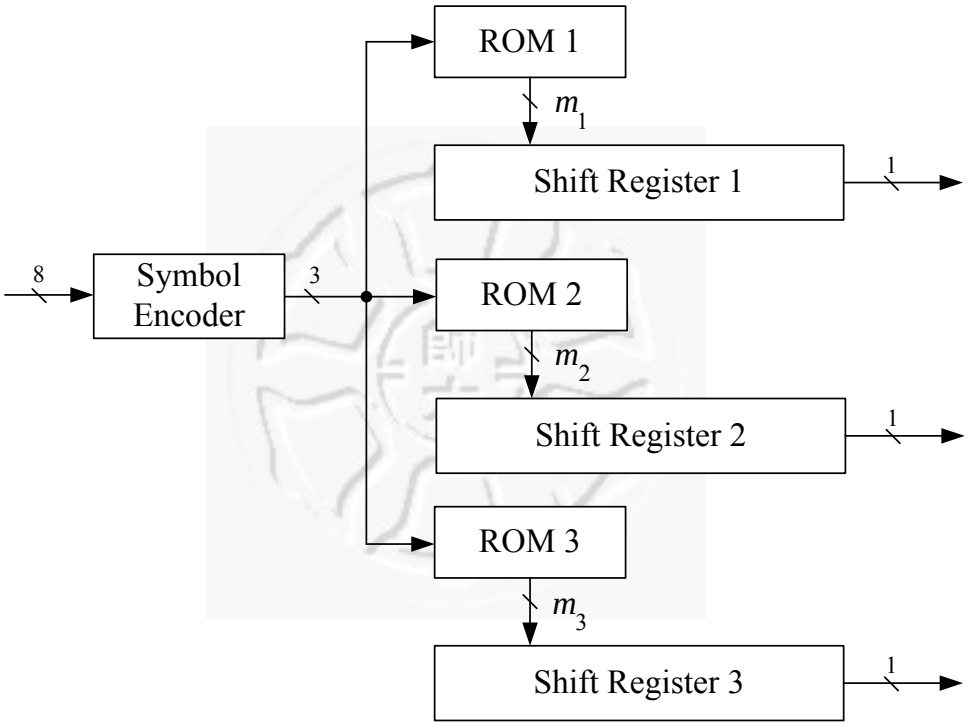

Figure 3.4: The sharing of the same symbol encoder by three different Snort rules. Each character is also assumed to be an ASCII. All the Snort rules use the same alphabet comprised of 7 symbols.

partition use the same set of symbols. Hence, all the rules in the same partition share the same symbol encoder as shown in Figure 3.4.

3.2 High Throughput Module Circuit

The basic module circuit shown in Figure 3.2 only process one character at a time. Actually, the throughput of the NIDS can be further enhanced by processing q characters at a time. This can be achieved by grouping q consecutive characters into a simple symbol. Without loss of generality, we group 2 characters into one new symbol x_i. Therefore, the total new symbol $x_1, ..., x_{|\Omega|}$ forms a new alphabet

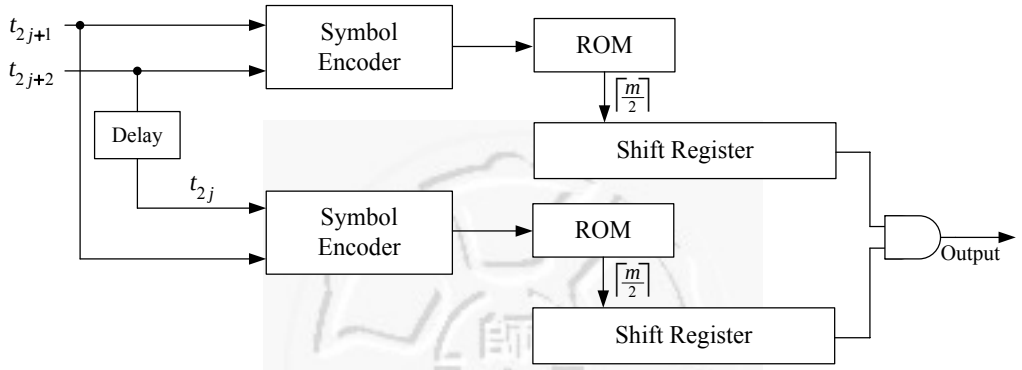

Figure 3.5: The structure of a high throughput module circuit processing two characters at a time ($q = 2$) with two single-port ROMs.

Ω.

Based on the alphabet Ω, we group two symbols p_{2i-1} and p_{2i} into u_i. Moreover, the pattern P can be expressed as $u_1 u_2 ... u_{\lceil m/2 \rceil}$. When m is even, the symbol $u_{\lceil m/2 \rceil}$ is composed of p_{m-1} and p_m. Otherwise, when m is odd, the symbol $u_{\lceil m/2 \rceil}$ is comprised of p_m and ϕ, where ϕ signifies "don't care", and can be any character in Σ. For each new symbol x_k, we can also define a bit vector X_k containing $\lceil m/2 \rceil$ elements. The i-th element of X_k is shown as follows.

$$X_k[i] = \begin{cases} 0, & \text{if } x_k = u_i, \\ 1, & \text{otherwise.} \end{cases} \tag{3.1}$$

The content of the ROM incoporates $X_1, ..., X_{|\Omega|}$ can be used for shift-or operations. In this case, ROM contains $|\Omega|$ entries which is equivalent to $|\Sigma|^2$. In addition, each entry occupies $\lceil m/2 \rceil$ bits. Therefore, a larger ROM should be adopted for modules. In order to reduce the ROM size, a symbol encoder similar as Figure 3.3 is also utilized.

The string matching operation ending at j with the alphabet Ω is identical to the operations ending at either $2j$ or $2j + 1$ with the alphabet Σ. It is essential to operate the string matching ending at every position of the source with the alphabet Σ. Hence, we employ two shift registers within the modules as shown in Figure 3.5. One of them is for even positions, and the other of them is for

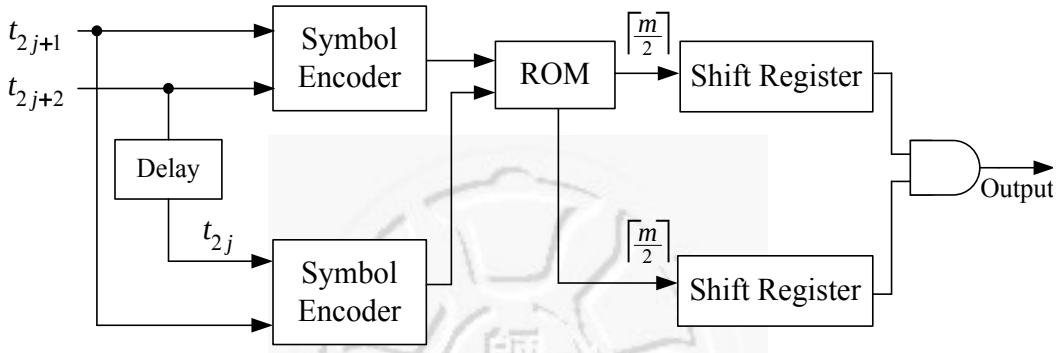

Figure 3.6: The structure of a high throughput module circuit processing two characters at a time ($q = 2$) with a shared dual-port ROM.

odd positions. Furthermore, since each entry of ROM contains only $\lceil m/2 \rceil$ bits, shift registers with $\lceil m/2 \rceil - 1$ flip-flops and OR gates with $\lceil m/2 \rceil$ are sufficient for the operations. Therefore, total flip-flops in the high throughput circuit is $2\lceil m/2 \rceil - 2$. We can observe from the result that the total flip-flops consumption is still less than the basic module circuit.

To perform the string matching operations ending at the *even* positions of the source with Σ, two characters t_{2j-1} and t_{2j} are then fetched to the ROM at the same time during the clock cycle j. In addition, during the clock cycle $j + 1$, characters t_{2j+1} and t_{2j+2} are also delivered to the ROM concurrently for shift-or operations.

The shift-or operations at the *odd* positions of the source can be performed in the similar manner. As shown in Figure 3.5, t_{2j} and t_{2j+1} can be obtained from the characters t_{2j-1}, t_{2j}, t_{2j+1} and t_{2j+2} during the clock cycle j and $j + 1$ in *even* positions through delaying and broadcasting operations. Consequently, the shift-or operations at even and odd positions share the same input as shown in the figure.

As shown in Figure 3.5, two identical ROMs read concurrently for each rule. The storage overhread can be further reduced by utilizing the dual-port ROM.

Table 3.1: Analysis of the proposed architecture with and without symbol encoder, where the number of characters for pattern matching is 1568 characters.

Design	Throughput (Gb/s)	Logic elements /char	Memory bits	Operating frequency (MHz)
Without symbol encoder	2.57	0.97	387,584	321.03
With symbol encoder	2.57	0.99	25,738	321.03

The dual-port ROM permits the same memory block to be shard by two concurrent reads, as illustrated in Figure 3.6. Altera Stratix FPGA devices support the dual-port ROM realized by the M4K blocks, where a true dual-port mode supporting any combination of dual-port operations: two reads, two writes, or one read and one write [2]. The utilization of these embedded memory blocks is extremely helpful for implementing our proposed circuits achieving both high throughput and low area cost.

3.3 Experimental Results

This section provides the experimental result of the proposed architecture. As shown in Table 3.1, it analyzes the throughput, logic elements per character, memory bits and operating frequency of the proposed circuits with and without the symbol encoder. The throughput denotes the maximum number of bits per second the circuit can process. For simplicity, the circuits processing one character at a time only is taken into consideration in the table. Altera Quartus II is the tool we adopted for circuit synthesization. The target FPGA device for the implementation is Stratix EP1S40.

As shown in Table 3.2, it compares the throughput, logic elements per character, memory bits and operating frequency of the proposed circuits with $q = 1$ and $q = 2$. Since the circuit with $q = 2$ processes two characters at a time, it has higher throughput than the circuit with $q = 1$ processing one character at a time

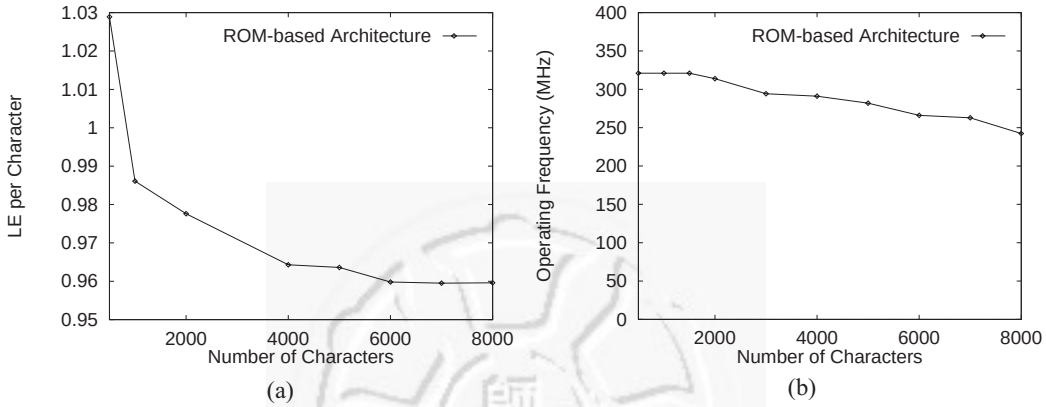

Figure 3.7: The performance of the proposed circuit with $q = 1$ for various rule sets with sizes ranging from 500 characters to 8000 characters (a)LE per character (b)Operating frequency

Table 3.2: Comparisons of the proposed architecture with $q = 1$ and $q = 2$, where the number of characters available for pattern matching is 1568 characters.

Design	Throughput (Gb/s)	Logic cells /char	Memory bits	Operating Frequency (MHz)
$q = 1$	2.57	0.99	25,738	321.03
$q = 2$	5.14	1.09	40,768	321.03

only. On the other perspective, we can also discover that the circuit with $q = 2$ has slightly higher amount of logic elements per character because of the complex symbol encoders. In addition, for $q = 2$, the string matching operations for each rules requires two independent ROMs as illustrated in Figure 3.5. Therefore, the number of memory bits consumed by the circuit with $q = 2$ is much more higher than the circuit with $q = 1$.

Figure 3.7 demostrates 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 utilized to reduce the storage size of the ROM. Moreover, different rules will share the same symbol encoder for reducing the area cost for the FPGA implementation.

Table 3.3: 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

Observed from Figure 3.7, the operating frequency of the proposed circuit is stable over a wide range of the rule set sizes. Furthermore, the average number of LEs per character decreases as the size of rule set increases. The reason is that the area cost of the symbol encoder reduces as the number of rules sharing the symbol encoder grows. To be more specificlly, when the rule set size reaches 8000 characters, the average number of characters becomes only 0.95 LE per character.

Table 3.3 compares the throughput, logic elements per character, memory bits and operating frequency of the proposed circuits for various configurations. Within the table, the circuits process two characters at a time with the rule set size 1568 characters. Since the alphabet size is 2^{16} for $q = 2$, ROMs for each rule occupies 2^{16} entries when the symbol encoder is not utilized. Hence, the total amount of memory bits is 102.76M bits as for the rule set size 1568 characters. Owing to large amount of embedded memory bits required for pattern saving, it is extremely impossible to implement the circuit by the existing FPGA devices. As shown in Table 3.3, the exploitation of the symbol encoder significantly reduce the memory bits for the ROM implementation (from 102.76M bits to 40.76K bits). Nonetheless, without the sharing of the symbol encoder by different rules, the number of logic element per character is 1.74. With the symbol encoder shared, the area cost can be reduced to 1.09 LEs per character. In addition, the clock rate of the circuit with the symbol encoder sharing can achieve up to 321.03MHz, which is significantly higher than the circuit without symbol encoder

sharing. As shown in Figure 3.6, when the ROM is also shared by string matching operations ending at even and odd positions, the number of memory bits can be significantly reduced by half (from 40,768 bits to 20,826 bits). As for the Altera Stratix FPGA devices, the shard ROM is implemented by true dual-port ROMs, which are supported only by M4K embedded memory blocks. On the contrary, the single-port ROM can be implemented by embedded memory blocks M512 with faster speed. Therefore, the proposed circuit with ROM sharing operates slower clock rate as compared with the circuit without ROM sharing, where ROMs are implemented by M512.