Chapter 2

Background

In this chapter, several existing works of hardware-based string matching systems for network intrusion detection systems are described. Existing research focuses on efficiency, hardware resource, and throughput of the architecture. The approach is comprised of two basic catagories: regular expression [5, 7, 9] and Content Addressable Memory (CAM) [8, 12]. In the final section of the chapter, shift-or algorithm we adopted for the circuit will be discussed.

2.1 Regular Expression

One popular approach for FPGA is based on regular expression. The regular expression approach is a low area cost solution, but does not achieve high throughput since it is extremely difficult to process more than one character at a time. The main purpose of the regular expression approach is to form a finite automata to search for the string. The finite automata can be further divided into nondeterministic finite automata (NFA) and deterministic finite automata (DFA).

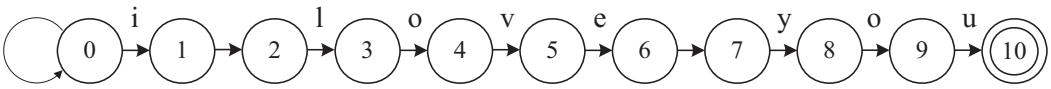

Figure 2.1: Regular Expression

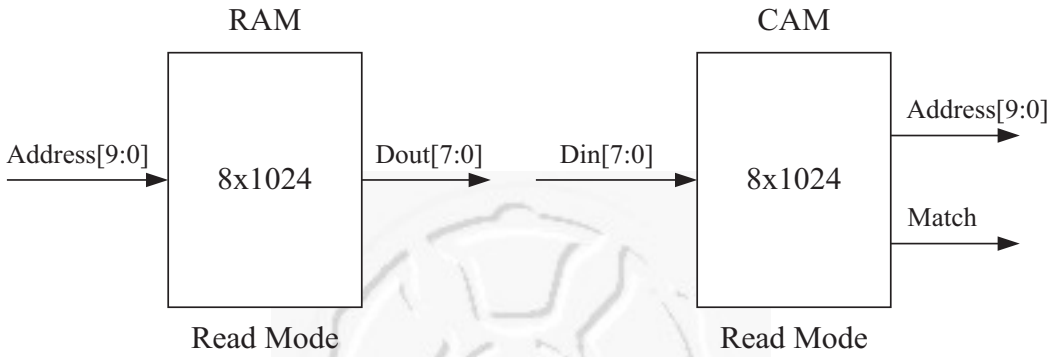

Figure 2.2: Comparisons between RAM and CAM

NFA can be represented by a directed graph, in which the nodes are the states and the labeled edges are the transitions. Each transition is denoted by a character or null character ϵ. A regular expression of m characters maps to $O(m)$ nodes. Constructing an NFA with $O(m)$ nodes takes $O(m)$ time and requires $O(m)$ memory. NFA processes each character in $O(m)$ time.

DFA is similar to NFA but DFA cannot permit multiple transitions over the same character and transitions over null character ϵ. A regular expression of m characters maps to $O(2^m)$ nodes. The total construction time with $O(2^m)$ nodes is $O(2^m)$ and DFA requires $O(2^m)$ memory. Although DFA need more memory than NFA, the merits of DFA is that DFA need less time to process characters. Moreover, DFA is able to process each character in $O(1)$ time.

Figure 2.1 shows an example of the regular expression approach. Input data is shifted through one character at a time. The state machine generates eleven states. The state represented by zero is considered an idle state. As characters are shifted in, the transitions are scanned. The finite state machine will move to subsequent state if the transition is matched. In the end, a matched string is found until the final state is reached.

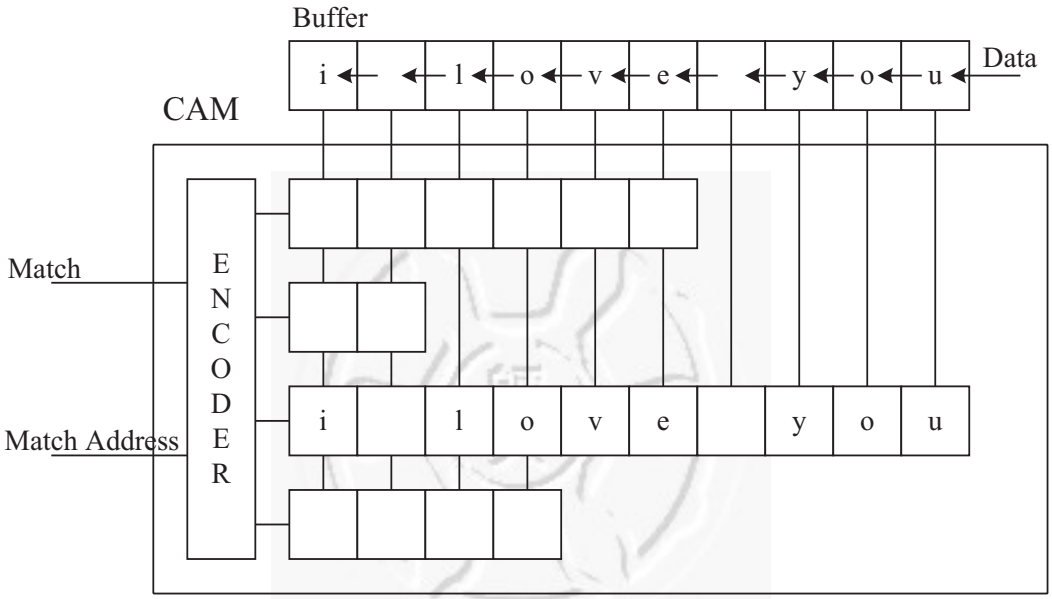

Figure 2.3: Comparisons between RAM and CAM

2.2 Content Addressable Memory (CAM)

Another alternative approach for FPGA implementation is to use the content addressable memory (CAM). CAM permits multiple input characters to be searched in parallel. This may effectively enhance the throughput of the architecture at the cost of higher area cost.

Actually, CAM circuits are similar with Random Access Memory (RAM). In RAM architecture, address is provided to hardware at first. Then, data at the dedicated address is returned. In CAM architecture, data is firstly supplied to the hardware. Each memory cell is accessed in parallel and compared to the data value. If the data value is found in the memory locations, a match signal is triggered and the address is also returned. Figure 2.2 shows the comparisons between RAM and CAM.

The conventional CAM architecture is illustrated in Figure 2.3. Each locations within the buffer is connected to all the horizontal square lines. The horizontal square lines called matching units contain strings for matching. When the input data is shifted into buffer, the buffer will broadcast all the data into matching

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

(a)

i \ 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
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)

Figure 2.4: 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).

units. When the data is matched, a matched signal and the match address is returned.

2.3 Shift-or Algorithm

In this thesis, we adopt the shift-or algorithm as the central idea to perform the architecture. The shift-or algorithm was proposed for exact string matching [3] by Baeza-Yates and Gonnet. To explain the shift-or algorithm in much more detail, we assume that we are searching for a *pattern* P comprised of $p_1p_2...p_m$ within a large *text* T with $t_1t_2...t_n$. Pattern P and text T contains the same alphabet $s_1, ..., s_{|\Sigma|}$ of Σ. R_j is a bit vector that indicates the conditions whether the prefix of pattern ended at j matches the text or not. $R_j[i]$ is set to zero if the first i characters of the pattern exactly match the last i characters up to j in the text. To elaborate on the concept, the transition from R_j to R_{j+1} can be expressed by the equation as follows.

$$R_{j+1}[i] = \begin{cases} 0, & \text{if } R_j[i-1] = 0 \text{ and } p_i = t_{j+1}, \\ 1, & \text{otherwise,} \end{cases} \tag{2.1}$$

The initial conditions of the equation are given that $R_0[i]$ is 1, where $i = 1, ..., m$. $R_j[0]$ is 0, where $j = 0, ..., m$. The equation can be implemented by

only shift and OR operations. In order to prove the scenario, each symbol s_k corresponds to a bit vector S_k is defined. A bit vector $S_k[i]$ can be expressed as follows.

$$S_k[i] = \begin{cases} 0, & \text{if } s_k = p_i, \\ 1, & \text{otherwise.} \end{cases} \tag{2.2}$$

Assume t_{j+1} is equal to s_c, we can rewrite eq.(2.1) based on eq.(2.2) as follows.

$$R_{j+1}[i] = R_j[i-1] \; OR \; S_c[i], i = 1, ..., m. \tag{2.3}$$

We can clearly see now the transition from R_j to R_{j+1} is no more than a shift of R_j and an OR operation with S_c. Figure 2.4 shows an example of the exact string matching based on shift-or algorithm, where the pattern is *aab* and the total symbol is a, b and c. In this example, text is *acaab*. Therefore, symbols s_k are a, c, a, a and b for $j = 1, 2, 3, 4$ and 5, respectively. The bit vector S_c associated with s_c for each j can be found from the table shown in Figure 2.4(a). Given S_c and R_{j-1}, the R_j can be computed by eq.(2.3), as shown in Figure 2.4(b). Notice that, when $j = 5$, it can be found from Figure 2.4(b) that $R_j[3] = 0$. In consequence, one occurrence of P is found when $j = 5$.