ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 4, Issue 12, December 2017

Synthesis of Aho-Corasick Algorithm for Network Processor

[1] Mrs. Neha Jain, [2] M.K. Jain

Abstract **- Network processor is an Application Specific Instruction Set Processor for networking application. In this paper we present the hardware implementation of Aho-Corasick algorithm for a network processor. Aho-Corasick algorithm is a pattern searching algorithm. This algorithm can be used to perform ip-lookup, in intrusion detection system etc. Much work has been done in this area, yet there is still a significant space for improvement in efficiency, flexibility, and throughput. In this paper we present the profiling data of software implementation of Aho-Corasick algorithm. After this we present the hardware implementation of the algorithm. In this paper we represent the complete source code. We implement the code using hardware description language like Verilog in Xilinx ISE. Total memory usage of this synthesis is 289108 kilobytes. Total power supply for this synthesis is only 274.02 mW. Only 86 Slice Flip Flops out of 55296 and 257 4 input LUTs out of 55296 are used in the synthesis of Aho-Corasick algorithm. Only 1% Slice Flip Flops and LUTs are used in this synthesis. Thus the device utilization of this implementation is also excellent..**

Index Terms:--- **Aho-Corasick algorithm, Kcachegrind, Masiff Visualizer, Network Processor Pattern Searching, Profiling.**

I. INTRODUCTION

A network processor is an Application Specific Instruction Set Processor for the networking application. Pattern matching, key lookup, data bit field manipulation, queue management are the optimized features or functions of a network processor. In this paper we present the hardware implementation of Aho-Corasick algorithm for a network processor. Aho-Corasick algorithm is a pattern searching algorithm. This algorithm can be used to perform ip-lookup, in intrusion detection system etc. In [1] author presented a hardware software co-design of Aho-Corasick algorithm in Nios II soft-processor and a study on its scalability for a pattern matching application. In [2] author proposed a reconfigurable hardware implementation for pattern matching using Finite State machine (FSM). In [3], [4], [9] authors presented hardware implementation of string matching algorithms. In [5] author proposed a hardware-efficient string matching architecture using the brute-force algorithm. In [6] author presented a fast pattern matching in strings. In [12] author proposed a Network processors (NPs) for active networks (AN).

II. PROFILING OF SOFTWARE IMPLEMENTATION OF THE ALGORITHM

Graph displayed in fig.1 is generated using Masiff Visualizer. Masiff shows the memory consumption at different-different snapshots. It takes snapshots at different time intervals. The bars shown by':' are normal snapshots. It means small amount of information is collected at these points. The bar shown by '@' are detailed snapshots. In the following graph 76 snapshots has been taken out of which [1, 12, 22, 25, 55, 63, 64, 70, 71, 72(peak)] are detailed snapshots. The bar showed by '#' is a peak snapshot. It shows the heighest memory consumption at this point.

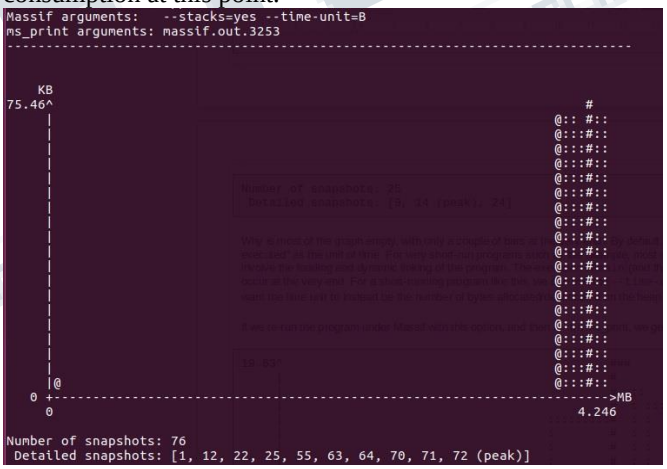


Fig. 1: Memory Consumption at Different Point of Program Execution

Fig.2 shows an allocation tree. It indicates exactly which piece of code is responsible for allocating heap memory. We read allocation tree from top to down. This allocation tree shows that 73,728B of useful heap memory has been allocated and arrows show that this is from different code location. Function call_init_part0 is using 72,704B out of 73,728B memory, which in turns used by _dl_init(). 1,024B memory out of 73,728B memory is used by IO_file_doallocate().

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 4, Issue 12, December 2017

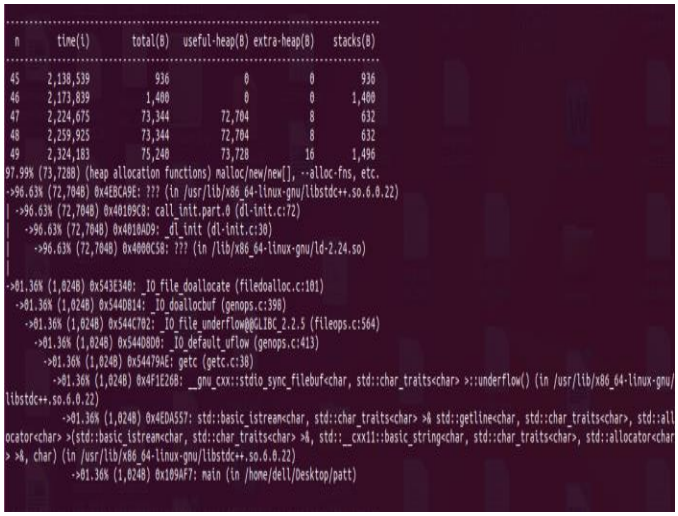
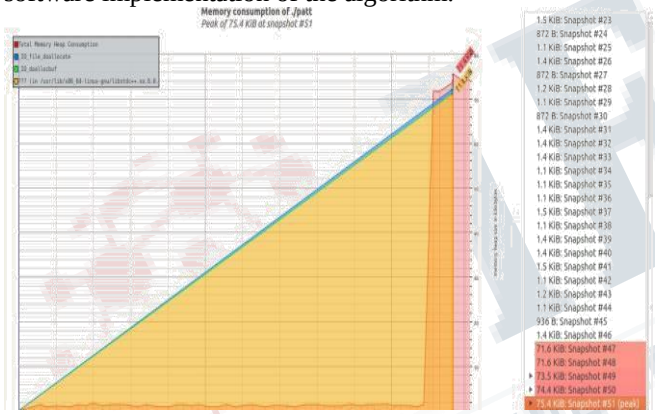


Fig. 2 Snapshot with detail Information Recorded

Fig 3 shows the peak memory consumption of the software implementation of the algorithm.



In fig. 4 we can see that 'inclusive' cost for findWords() is 3.95 and 'Self' cost is 0.04. Similarly 'inclusive' cost for matching () is 2.94 and 'Self' cost is 0.38. findWords() and matching() are the functions declared and defined in software implementation of algorithm. If the 'inclusive' cost is high and self cost is low than we have to optimize the function. Thus to optimize these function we will switch to hardware implementation.

Incl.	Self	Called	Function
3.85	0.29	91	_dl_fixup
3.72	0.01	1	findWords(std::__cxx11:.
3.30	0.15	71	_dl_runtime_resolve_ss.
2.96	0.38	1	matching(std::__cxx11::..
2.93	0.01	1	_dl_init
2.93	0.02	7	call_init.part.0

Fig. 4 Inclusive and Self Cost of Functions

III. IMPLEMENTATION OF AHO-CORASICK ALGORITHM IN XILINX ISE

Main working principle of this module is to find specific strings in data input. It has following:
Inputs
clk – 1 input, reset – 1 input, start - 1 input, stop -1 input data – 8 parallel inputs.
Outputs:
Shot – 1 output, hot– 1 output, hyp– 1 output, shy -1 output, shot_position – 8 parallel outputs, shot_position_end - 8 parallel outputs, hot_position - 8 parallel outputs, hot_position_end - 8 parallel outputs, shy_position - 8 parallel outputs, shy_position_end - 8 parallel outputs, hyp_position - 8 parallel outputs, hyp_position_end - 8 parallel outputs
In this module we have set all possible letters which we have in our strings (shot, shy, hyp , hot -- all possible letters in this strings are s, h, t, o, y, p) as a parameters. These parameters we use in state machine to compare with input letters of random word, for input text we have "data", because module has a 8 inputs we input letter(which is 8 bits) one by one of word after each clock and in that process a module compare with set parameters (s, h, t, o, y, p), inside of a module are settings for each of strings (shot, shy, hyp , hot) will result output with position and string who is contained in input word.
Settings are set for each string and it is using Aho-Corasick algorithm which means that if we have on input a "hyp" word for example, module will recognize h letter on input and he will set two possible strings hyp and hot, after next clock we have y so only potential string is hyp, string hyp is using final state machine FSM(fig.6) and has a condition, if is going from h to y and then to p then it will be set "hyp" output, and position of string in input word hyp_position and hyp_position_end, it is possible to have word on input which contain more than one or all of our strings. Position is starting to be calculated from the moment we activated "start", before running all we need to set "reset" to reset module. "stop" input has a function to put module in IDLE condition in which case he remember outputs but "data" input is not accepted, to continue we set "start" and all output will be reset and we can continue with "data" input.

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 4, Issue 12, December 2017

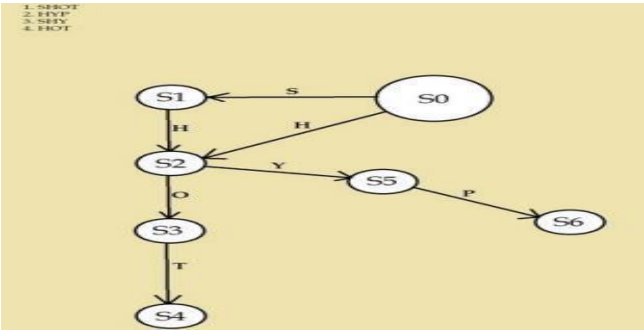


Fig. 5 State Table

Table 1 Transition Table

S. No	Current State	Input Character	Next State
1	S0	S	S1
2	S0	H	S2
3	S2	Y	S5
4	S5	P	S6
5	S1	H	S2
6	S2	O	S3
7	S3	T	S4

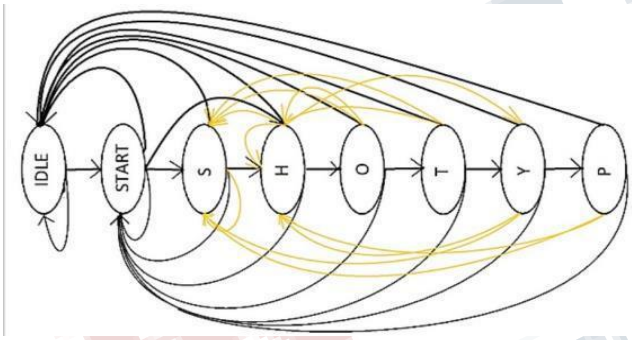


Fig. 6 FSM

Veriolog code for the implementation is as follows:

A. Wordfinder.v

```
module wordfinder (clk, reset, start, stop, data, shot, shot_position, shot_position_end, hot, hot_position, hot_position_end, shy, shy_position, shy_position_end, hyp, hyp_position, hyp_position_end);
input clk; input reset; input start; input stop; input [7:0] data; output shot; output hot; output shy; output hyp; output [7:0] shot_position; output [7:0] shot_position_end;

output [7:0] hot_position; output [7:0] hot_position_end; output [7:0] shy_position; output [7:0] shy_position_end; output [7:0] hyp_position; output [7:0] hyp_position_end; wire [7:0] data_in; assign data_in = data; //DATA PARAMETERS

parameter S_DATA = 8'h53; parameter H_DATA = 8'h48; parameter O_DATA = 8'h4f; parameter T_DATA = 8'h54; parameter Y_DATA = 8'h59; parameter P_DATA = 8'h50;

//LENGTH PARAMETERS
parameter SHOT_LENGTH = 4
-1;

parameter HOT_LENGTH  = 3 -1;
parameter SHY_LENGTH  = 3 -1;
parameter HYP_LENGTH  = 3 -1;
//STATE PARAMETERS

parameter IDLE = 8'b00000001;
parameter START =
8'b00000010;

parameter S = 8'b00000100; parameter H = 8'b00001000; parameter O = 8'b00010000; parameter T = 8'b00100000; parameter Y = 8'b01000000; parameter P = 8'b10000000; //REGISTERS

reg [7:0] state_next; reg [7:0] state_reg;
reg [7:0] counter_next; reg [7:0]
counter_reg;

reg shot_start_next;    reg shot_start_reg;

reg shot_next;          reg shot_reg;

assign shot = shot_reg |
shot_next; reg [7:0]
shot_position_next; reg [7:0]
shot_position_reg;

assign shot_position = (state_reg == T) ?
shot_position_next : shot_position_reg;

reg [7:0]
shot_position_end_next; reg
[7:0] shot_position_end_reg;

assign shot_position_end = (state_reg == T) ?
shot_position_end_next :
shot_position_end_reg; reg shy_start_next; reg
shy_start_reg;
reg shy_next;          reg shy_reg;

assign shy = shy_reg | shy_next;
```

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 4, Issue 12, December 2017

```
reg [7:0] shy_position_next; reg [7:0] shy_position_reg;
assign shy_position = (state_reg == Y) ? shy_position_next :
shy_position_reg; reg [7:0] shy_position_end_next;
reg [7:0] shy_position_end_reg;
assign shy_position_end = (state_reg == Y)
?shy_position_end_next : shy_position_end_reg;

reg hot_next;              reg hot_reg;
assign hot = hot_reg | hot_next; reg [7:0]
hot_position_next; reg [7:0] hot_position_reg;
assign  hot_position = (state_reg == T) ?
hot_position_next :
hot_position_reg;
reg [7:0]
hot_position_end_next; reg
[7:0] hot_position_end_reg;
assign hot_position_end = (state_reg == T) ?
hot_position_end_next : hot_position_end_reg;
reg hyp_next; reg hyp_reg;
assign hyp = hyp_reg |
hyp_next; reg [7:0]
hyp_position_next; reg [7:0]
hyp_position_reg;
assign hyp_position = (state_reg == P) ? hyp_position_next :
hyp_position_reg;
reg [7:0]
hyp_position_end_next; reg
[7:0] hyp_position_end_reg;
assign hyp_position_end = (state_reg == P) ?
hyp_position_end_next : hyp_position_end_reg;
initial begin
$display("WORDFINDER");
end
//FLIP FLOPS
always @(posedge clk) begin
 if(reset==1) begin
state_reg = IDLE; counter_reg = 0;
shot_start_reg = 0; shy_start_reg =
0; shot_reg = 0; shy_reg = 0;
hot_reg = 0; hyp_reg = 0;
shot_position_reg = 0;
shy_position_reg = 0;
hot_position_reg = 0; hyp_position_reg = 0;
shot_position_end_reg = 0; shy_position_end_reg =
0; hot_position_end_reg = 0; hyp_position_end_reg
= 0; end else begin
state_reg = state_next; counter_reg =
counter_next; shot_start_reg = shot_start_next;
shy_start_reg = shy_start_next;
shot_reg = shot_next; shy_reg = shy_next;
hot_reg = hot_next; hyp_reg = hyp_next;
shot_position_reg = shot_position_next;
shy_position_reg = shy_position_next;
hot_position_reg = hot_position_next;
hyp_position_reg = hyp_position_next;
shot_position_end_reg =  shot_position_end_next;
shy_position_end_reg = shy_position_end_next;
hot_position_end_reg = hot_position_end_next;
hyp_position_end_reg = hyp_position_end_next;
end
end
//CONTROL PATH and 1/2 DATA PATH
always @(*) begin
case(state_reg)
IDLE: begin
if(start == 1)
  state_next = START;
else
  state_next = IDLE; counter_next = 0; shot_start_next =
0;
  shy_start_next = 0;
end
START: begin
if(stop == 0) begin
  case (data_in)
  S_DATA: state_next = S; H_DATA: state_next = H;
  default: state_next = START;
  endcase
end else begin
state_next = IDLE;
end
counter_next = counter_reg +
1; shot_start_next =
shot_start_reg; shy_start_next
= shy_start_reg; end
S: begin
if(stop == 0) begin

case (data_in)
S_DATA: state_next = S;
H_DATA: state_next = H;
default: state_next = START;
endcase
end else begin
state_next = IDLE;
end
counter_next = counter_reg + 1;
if (data_in == H_DATA)
shot_start_next = 1;
else
shot_start_next = 0;
if (data_in == H_DATA)
shy_start_next = 1;
```

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 4, Issue 12, December 2017

```
else
shy_start_next = 0;
end
H: begin
if(stop == 0) begin
case (data_in)
S_DATA: state_next = S;  Y_DATA: state_next = Y;
H_DATA: state_next = H; O_DATA: state_next = O;
default: state_next = START;
endcase
end else begin
state_next = IDLE;
end
counter_next = counter_reg + 1;
if (data_in == O_DATA)
shot_start_next = shot_start_reg;
else
shot_start_next = 0;
if (data_in == Y_DATA)
shy_start_next = shy_start_reg;
else
shy_start_next = 0;
end
O: begin
if(stop == 0) begin
case (data_in)
S_DATA: state_next = S;  H_DATA: state_next = H;
T_DATA: state_next = T;  default: state_next = START;
endcase
end else begin
state_next = IDLE;
end
counter_next = counter_reg + 1;
if (data_in == T_DATA)
shot_start_next = shot_start_reg; else
shot_start_next = 0;
shy_start_next = shy_start_reg; end
T: begin
if(stop == 0) begin
case (data_in)

S_DATA: state_next = S;
H_DATA: state_next = H;
default: state_next = START;
endcase
end else begin
state_next = IDLE;
end
counter_next = counter_reg + 1;
shot_start_next = 0;
shy_start_next = shy_start_reg;
end
Y: begin
if(stop == 0) begin
case (data_in)
S_DATA: state_next = S;
H_DATA: state_next = H;
P_DATA: state_next = P;
default: state_next = START;
endcase
end else begin
state_next = IDLE;
end
counter_next = counter_reg + 1;
if (data_in == T_DATA)
shy_start_next = shy_start_reg; else
shy_start_next = 0;
shot_start_next = shot_start_reg; end
P: begin
if(stop == 0) begin
case (data_in)
S_DATA: state_next = S;
H_DATA: state_next = H;
default: state_next = START;
endcase
end else begin
state_next = IDLE;
end counter_next = counter_reg + 1;
shy_start_next = 0;
shot_start_next = shot_start_reg;
end  endcase // state_reg
end // always (*) CONTROL PATH and 1/2 DATA
PATH //OUTPUT MANAGE SHOT/HOT always @(*)
begin
case(state_reg)
IDLE: begin
if (start == 1) begin
shot_next = 0;      hot_next  = 0;
shot_position_next = 0;  hot_position_next  =  0;
shot_position_end_next = 0; hot_position_end_next = 0;
end else begin
shot_next = shot_reg;
hot_next  = hot_reg;
shot_position_next  = shot_position_reg;
hot_position_next  =  hot_position_reg;
shot_position_end_next  = shot_position_end_reg;
hot_position_end_next  =  hot_position_end_reg;
end
end
T: begin
if (shot_start_reg == 1)
shot_next = 1;
```

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 4, Issue 12, December 2017

```
else
shot_next = shot_reg;
if (shot_start_reg == 1) begin
shot_position_next = counter_reg - SHOT_LENGTH;
shot_position_end_next = counter_next; end else begin
shot_position_next  = shot_position_reg;
shot_position_end_next = shot_position_end_reg; end
hot_next  = 1;
hot_position_next = counter_reg - HOT_LENGTH;
hot_position_end_next = counter_next; end
default: begin
shot_next = shot_reg;
shot_position_next  = shot_position_reg;
shot_position_end_next = shot_position_end_reg;
hot_next = hot_reg;
hot_position_next = hot_position_reg;
hot_position_end_next = hot_position_end_reg; end
endcase end
//          always (*) OUTPUT MANAGE SHOT/HOT
//OUTPUT MANAGE SHY/HYP
always @(*) begin case(state_reg) IDLE: begin
if (start == 1) begin
shy_next  = 0;      hyp_next  = 0;
shy_position_next  =  0;    hyp_position_next  =  0;
shy_position_end_next  =  0;
hyp_position_end_next  =  0;
end else begin
shy_next  = shy_reg; hyp_next  = hyp_reg;
shy_position_next  =  shy_position_reg;
hyp_position_next  =  hyp_position_reg;
shy_position_end_next  =  shy_position_end_reg;
hyp_position_end_next  =  hyp_position_end_reg;
end
end
Y: begin
if (shy_start_reg == 1)
shy_next = 1;
else
shy_next = shy_reg;
if (shy_start_reg == 1) begin
shy_position_next = counter_reg - SHY_LENGTH;
shy_position_end_next = counter_next;
end else begin
shy_position_next  = shy_position_reg;
shy_position_end_next = shy_position_end_reg; end
hyp_next  = hyp_reg;
hyp_position_next = hyp_position_reg;
hyp_position_end_next = hyp_position_end_reg; end
P: begin
shy_next = shy_reg;
shy_position_next = shy_position_reg;
shy_position_end_next = shy_position_end_reg;
hyp_next = 1;
hyp_position_next = counter_reg - HYP_LENGTH;
hyp_position_end_next = counter_next; end
default: begin
shy_next = shy_reg;
shy_position_next = shy_position_reg;
shy_position_end_next = shy_position_end_reg;
hyp_next = hyp_reg;
hyp_position_next  =  hyp_position_reg;
hyp_position_end_next  =  hyp_position_end_reg;
end
endcase
end // always (*) OUTPUT MANAGE SHOT/HOT
endmodule // WORDFINDER
```

IV. RESULTS

Register level schematic view of the implementation is as follows:

Fig. 7 Register Level Schematic View

Total memory usage of this synthesis is 289108 kilobytes. Total power supply for this synthesis is only 274.02 mW. Only 86 Slice Flip Flops out of 55296 and 257 4 input LUTs out of 55296 are used in the synthesis of Aho-Corasick algorithm. Only 1% Slice Flip Flops and LUTs are used in this synthesis. Thus the device utilization of this implementation is also excellent. The detail report of device utilization can be seen in following table 2.

Table 2 Device Utilization

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 4, Issue 12, December 2017

Logic Utilization	Used	Available	Utilization
Number of Slice Flip Flops	86	55,296	1%
Number of 4 input LUTs	257	55,296	1%
Number of occupied Slice	130	27,648	1%
Number of bonded IOBs	80	489	16%
Number of BUFGMUXs	1	8	12%
Average Fanout of Non-Clock Nets	3.61		

The timing delay report can be seen in table 3.

Table 3 Timing Delay Report

Timing constraint: Default path analysis Total number of paths / destination ports: 68 / 64				
Delay:	11.872ns (Levels of Logic = 6)			
Source:	start (PAD)			
Destination:	shot_position<2> (PAD)			
Data Path: start to shot_position<2>				
Cell:in->out	fanout	Gate Delay	Net Delay	Logical Name (Net Name)
IBUF:I->O	12	0.715	1.120	start_IBUF (start_IBUF)
LUT3_D:I1->O	33	0.479	1.603	hot_position_end_next<0>11 (N3)
LUT4_L:I3->LO	1	0.479	0.123	shot_position_next<2>_SW0 (N10)
LUT4:I3->O	2	0.479	0.804	shot_position_next<2> (shot_position_next<2>)
LUT3:I2->O	1	0.479	0.681	shot_position<2>1 (shot_position_2_OBUF)
OBUF:I->O		4.909		shot_position_2_OBUF (shot_position<2>)

REFERENCES

[1] S.M. Vidanagamachchi, S.D. Dewasurendra, R.G. Ragel," Hardware software co-design of the Aho-Corasick algorithm: Scalable for protein identification 2013 IEEE 8th International Conference on Industrial and Information Systems, ICIIS 2013, pp.321-325, December 2013.

[2] N. I. Rafla, I. Gauba, "A Reconfigurable Pattern Matching Hardware Implement using On-Chip RAM-Based FSM," 53rd IEEE International Midwest Symposium on Circuits and Systems, August 2010.

[3] V. Dimopoulos, J. Papaefstathiou, D. Pnevmatikatost ," A Memory-Efficient Reconfigurable Aho-Corasick FSM Implementation for Intrusion Detection Systems," IC-SAMOS-2007, pp. 186-193, August 2007.

[4] G. F. Ahmed, N. Khare,"Hardware based String Matching Algorithms: A Survey," International Journal of Computer Applications, Vol. 8, February 201.

[5] C. C. Chen, S. D. Wang, "A Multi - character Transition String Matching Architecture Based on Aho - Corasick Algorithm", ICIC International, Vol 8, pp. 8367 -8386, 2012.

[6] S. Ahn, H. Hong, H. Kim, J. Ahn, "A Hardware - Efficient Multi - character String Matching Architecture Using Brute - force Algorithm", IEEE, ISOCC-2009, pp.464-467, 2009.

[7] D.E. Knuth, J.H. Morris and V.R. Pratt, "Fast pattern matching in strings," SIAM Journal of Computing, vol. 6, pp. 323–350, June, 1977.

[8] L. Salmela, J. Tarhio, J. Kytojoki, "Multi - Pattern String Matching with Very Large Pattern Sets", ACM Journal Algorithmic, Volume 11, 2006.

[9] M. Burrows, D. J. Wheeler, "A Block-sorting Lossless Data Compression Algorithm," SRC Research Report, May, 1994.

[10] E. Fernandez, W. Najjar, S. Lonardi, "String Matching in Hardware using the FM – Index" , IEEE International Symposium on Field - Programmable Custom Computing Machines , 2011.

[11] A. Kind, R. Pletka, M. Waldvogel "The Role of Network Processors in Active Networks, IFIP International Working Conference on Active Networks, pp. 20-31, 2003.