IOSR Journal of VLSI and Signal Processing (IOSR-JVSP)
Volume 8, Issue 4, Ver. I (Jul. - Aug. 2018), PP 07-14
e-ISSN: 2319 – 4200, p-ISSN No. : 2319 – 4197
www.iosrjournals.org

An Optimized Multi-Character Pattern Matching Circuit For Network Intrusion Detection Systems

Bala Modi[1], Gerald Tripp[2]
[1] *(Department Of Mathematics, Faculty Of Science, Gombe State University, Gombe Nigeria)*
[2] *(School Of Computing, Faculty Of Science, University Of Kent, CT2 7NF, Canterbury, United Kingdom)*
Corresponding Author: Bala Modi1

Abstract: *There are various kinds of Network attacks often identifiable by the patterns of data they contain. More complex regular expressions that express these patterns need to be matched at a very high speed. Most hardware-based approaches build the equivalent automata using minimal hardware resources to detect pattern variations. This paper explains the design, structure, and suitability of an optimized hardware-based automata implementation called Equivalence Class Direct Table Synthesis Nondeterministic Finite Automata (ECD$_R$TS-NFA). The optimized approach described in this paper builds upon the earlier published version called Equivalence Class Descriptor Nondeterministic Finite Automata (ECD-NFA). The ECD$_R$TS-NFA also uses an Equivalence Classification (EC) technique. However, the ECD$_R$TS-NFA approach utilizes a newer form of table compression for its compressed ECs, called Equivalence Class Descriptors (ECDs). The ECDs then used to match against multi-character strings rather than the initial single character approach implemented in the ECD-NFA design. The optimized technique implemented in the ECD$_R$TS-NFA further improves the matching speed of the design, while at the same time significantly reducing the overall resources required. This is achieved by taking full advantage of the Field Programmable Gate Array (FPGA) technology used for the hardware implementation. The design further provides higher throughput and support for quick updates, and clocks at 385.78 MHz, with a maximum throughput value of 12.34 Gigabits per second (Gbps), depicting a 3.35% improvement over the next best rival design in this paper.*

Keywords: *ECDs, ECD-NFA, FPGA, LUTs, Throughput.*

Date of Submission: 16-07-2018 Date of acceptance: 31-07-2018

I. Introduction

Most corporate network attacks target the privacy and confidentiality of both network clients and confidential documents. These attack patterns could be in any form ranging from *spams, bugs, denial-of-service (DoS)* attacks to *malicious software* such as: *viruses, worms, Trojan horse, spyware, hybrid, droppers* and *blended threats* [1]. As such, it is imperative for pattern matching approaches to find predefined patterns in a wide range of data streams [2] due to the vast number of data streaming through a network on daily basis. A pattern set composed of thousands of patterns could grow to enforce new policies related to security issues [3].

The matching processes performed could be regular expression or exact string matching. Exact string matching on a given packet can be performed during the process of deep packet inspection of the packet payload flowing into a given network [30], [32]. Exact string [16] matching techniques are weak against current patterns of attacks which are mostly in form of regular expressions [31]. Popular and current software tools [3], [4], [5] now use *regular expressions* or simply termed 'regexps' to describe payload patterns [6]. Software solutions for regexp pattern matching have become inadequate in coping with the frequency of network attacks.

There are some attempts to optimize regular expressions before automation, in order to reduce the memory footprint. The approach by [33] uses JavaScript Object Notation (JSON) to optimize definitions with large and complex patterns. The approach by [34] is a parallel implementation based on a content-addressable memory (CAM). The approach iteratively compares the portions of an input traffic stream with the already stored character strings within the CAM, and in each of the search cycles, it concurrently compares the character strings stored within the CAM against the input traffic stream. How quickly the state definitions grow remains a question. However, alternative solutions to software approaches are the hardware-based regexps pattern matching designs, which are based on hardware technologies such as the: Application-specific Integrated Circuits (ASICs) [8], Graphic Processing Units (GPUs) [9], [10], and Field-programmable Gate Arrays (FPGAs) [11], [12], [13], and [14].

It is important to note that patterns matched by regexps can easily [7] be matched by an automaton, such as: Deterministic Finite Automata (DFA) or Nondeterministic Finite Automata (NFA). A variation of the former and the latter automata is called the *hybrid* DFA-NFA (hybrid-FA) [6]. Furthermore, with NFAs, each character to be matched is processed in O(n) time, and requires only O(n) memory. However, the NFA processing time could be reduced to O(1), but require O(n^2) memory [15] on FPGAs, and is achievable by exploiting it's fine-grained parallelism, which is a great advantage over microprocessors. NFA-based approaches are fast becoming popular because of their memory conservation and ability utilize the *re-configurability* [19] structure that current FPGAs have. The regexps utilized in this paper design are drawn from rulesets found in the popular Snort community rulesets [30], [3].

The EC technique described in [31] is retained for classifying all the input strings that have the same effect on the automata. The process also creates the relevant ECDs. The ECDs are then assigned and used to drive the automata. The matching process utilizes Block Random Access Memories (BRAMs) and is used to perform the software compression of all the raw data inputs into their various equivalence class vectors of next state transitions. The equivalence classes of vector states are then mapped to their respective ECDs accordingly [32]. This paper describes the novel table synthesis process of the compressed inputs on a target FPGA, which is built into the *ECD_RTS-NFA*. The matching process also incurs minimal logic circuit cost in comparison to the other related approaches. The *ECD_RTS-NFA* design is also quite suitable for implementation on any high speed network that is capable of performing multi-character matching. The most interesting thing about the *ECD_RTS-NFA* design is that it capable of matching 32-bit characters at time, which is a real improvement over the initial 8-bit characters matched by the previous *ECD-NFA* [31].

The remainder of this paper is organised thus: the related works is described and summarized in Section II. Section III describes the classification algorithm used for constructing the optimized ECD_RTS-NFA machine, as well as the process used in creating n-byte ECDs, with n=1,2,4. Section III also explains the overall structure of the design and the results of the preliminary evaluation obtained for the ECD_RTS-NFA design. Section IV compares and gives a brief analysis of the various related designs under consideration by using charts to evaluate the preliminary results obtained from Section III. Only FPGA-based implemented designs were considered and studied for comparison in this paper. Finally Section V discusses the conclusion and ideas for future work.

II. Related Works

The NFA logic described in [15] is a Finite State Machine (FSM) based approach that utilizes FPGA technology, and produces an output that is in form of a binary tree. Normally, a placed and routed netlist is built before generating *configuration bits* (bitstreams) at runtime in [31],[17]. The generated bitstream file is what is needed to program the FPGA device. In [15], while implementing NFAs as logic, it was realized that if all the source input Flip-Flops (FFs) to the destination input FFs [21] are on ϵ–transitions (epsilon transition), then the FFs can be eliminated without being implemented at all, and that could reduce the overall logic circuit size. This idea by [15] set the pace for several other approaches built upon reconfigurable [18][12] approaches.

The design by [23] resolved the problem of prefix sharing used to avoid unnecessary repeated searches, attributed directly to shared infix and postfix sub-patterns. The design memorizes the path that the trigger signal based on specific constraints suitable for both *exact* string matching and complex regexp matching. Also, the approach in [35] uses filters for the regexp query, and is classified into positive factor and negative factor. This was achieved by reviewing three typical positive factors, which include: prefix, suffix, and necessary factor so as to show that negative factors can collaborate with positive factors to significantly improve the filtering ability. A Perl Compatible Regular Expression (PCRE) compiler that converts regexps from the Snort ruleset into PCRE *opcodes* was implemented in [24]. The opcodes are instructions for the software based PCRE engine defined in a file called *pcre_internal.h*, which is part of the PCRE Package. The compiler translates the PCRE opcodes into VHSIC Hardware Description Language (VHDL) codes necessary for parallel implementation in the FPGA. The design by [24] used a wider input bus through an SRAM interface, to increase the overall matching throughput.

The design by [7] is an automatic architectural optimisation approach which spatially stacks regexps matching circuits (REMs). It then forms multiple character matching (MCMs) circuits. The MCMs are then grouped into clusters and marshaled onto a two dimensional staged and pipelined structure. The structure is aimed at improving the overall design clock speed [31]. However, the challenge with the architecture designed in [7] is that the process of distributing and buffering the character matching signals was initially error-prone and difficult to implement manually. To address the problem, the approach proposed in [25] used a heuristic that automatically marshaled *k*-REMs with total *N*-states into *p*-pipelines. The process selects and executes a function that compares each generated character class within each REM. The comparison is done against those that were previously collected in the BRAM, whenever a REM is to be added to an existing pipeline. The matching outputs [25] of each of the REMs are prioritized. The REM with higher priority is given to the lower-indexed pipelines and stages for the sake of efficiency. The phase-wise pipeline structure of the approach is very important to understanding how the ECD_RTS-NFA design in this paper operates.

The concept of classification of character input strings [31] for driving the *ECD_RTS-NFA*-based automata at a more appreciable clock rate is also very necessary in this paper. In addition, this paper design has further reduced the number of BRAMs used for storing ECDs. Also, the function generators also termed look-up tables (LUTs) were synthesized to ease the translation of inputs for matching patterns of attacks [32]. This unique form of direct table synthesis technique, synthesizes the memory blocks into pure logic using only LUTs used for translating 2-byte and 4-byte tables of ECDs. Furthermore, the designs in [26], [27], [28] and [22] utilized a similar concept but implemented it for a DFA approach instead. The ECD_RTS-NFA design like the ones in [32] and [28] takes a given number of regexps and generates the equivalent VHDL codes. The generated VHDL codes are essential for easing re-the configuration process of the design. Although, the higher number of characters (32-bits) consumed by ECD_RTS-NFA machine has increased the throughput of matching, it was achieved at the cost of lower operational clock speed, which is a trade-off in such approaches.

III. The ECD_RTS-NFA Classification Algorithm

The concept of equivalence classification can best be understood based on the definitions discussed in [29] and detailed in [31]. The Algorithm 1 discussed in [31] has been modified such that the *m-character* class of next state vectors, where *$0<m<1$* is increased to *$0<m<4$* to produce n-ECDs. Algorithm 2 and Figure 1, further describes the unique process of synthesizing memory into pure logic as LUTs.

Algorithm 1: Construction of an *n-ECD* vectors of next states [31]

INPUT: An *n-state, m-character class* ECDs. The input state is *s*.
OUTPUT: An *n-state ECD-NFA* with the associated *multi-byte* table of *compressed* ECDs.
BEGIN

i. Read and parse the regexps to be constructed into the equivalent ECD_RTS-NFA.
ii. For $\forall$ *i* < *n*, where *i* = 0,1,2,3....*n*-1, and *n* is the total number of states in the NFA. If the transition (link) from state s_i is a self-transition from state s_i to itself upon consuming a non-empty character, remove all such self-transitions.
iii. For $\forall$ $i < n$, $j < n$ and $k < n$, if the output of state s_i connects to the state inputs of some state s_j upon consuming an empty string (ϵ), remove all such transitions $t_{i,j}$ linking state s_i to s_j. Create a new transition that connects state s_i to states s_j and s_k where $s_k > s_j$ on a non-empty input.
iv. For $\forall$ $i < n$, $j < n$ and for each transition $t_{i,j}$ from a state s_i to a state s_j, scan through. Store all next states transited to on the same input, into a set of next states. Store all the different sets of next states into a single vector of sets of next states and assign a single input character class descriptor to them.
v. Assign to each classified inputs created in (iv) ECDs, which are the class descriptors. The ECDs now represent the sets of vectors of next states for all character classes that trigger transitions from a state s_i to a state s_j, where $i < n, j < n$.
vi. Repeat steps (i) - (v) for $\forall$ s_i, $i < n$ and store all the sets of vectors of next states in a list of state vectors for $\forall$ states s_i in the ECD_RTS-NFA.
vii. Once step (vi) is completed, the process of building the compressed table of ECDs begins. The process first performs the cross product computation of any two sets of vectors of next states v_i and v_j $\forall$ $i < n, j < n$ contained within the list of state vectors stored in (vi). Subsequently, all the similar vectors are merged to become a single vector. Recursively performing step (vi) – (vii) generates a 4-byte table of ECDs two 2-byte tables.
viii. Finally, exit the process after step (vii) and generate the VHDL file for the ECD_RTS-NFA. The file is then uploaded to the XST VHDL synthesis tool for synthesis and implementation.

END

A. ECD_RTS-NFA Circuit Block

The design classification algorithm was described in [31] in more detail. The algorithm generates the required inputs (ECDs) using Algorithm 1 which constructs vector of next states. The vectors of next states were formed and classified accordingly into 1-byte ECDs, 2-byte ECDs and 4-byte ECDs. For instance, the cross product between ECD 0 x ECD 1 = 3, causes the automata to transit to state 3 from state 1 on ECD input of 1 as seen in in the transition Table 1. Also, the cross product between ECD 1 x ECD 2 = ECD (1x2) = 4 as seen in in the transition Table 2 reflects the cross product between ECD (1x2) = 4. By recursively running the process again as explained in step (viii) of Algorithm 1 and merging the various state vectors seen in Table 2, the 4-byte ECDs are created as seen in Table 3 [31].

The ECDs in Tables 2 and 3 are then used to perform the table look up operation, which maps the 2-byte ECDs to 4-byte ECDs [31]. The vector of next states that were converted into the ECDs in Tables 2 and 3 were actually formed from the regexp "*/(a|b)*(cd)/*" and the process has been discussed in [31]. The tables are then synthesized respectively into logic using the Algorithm 2. The rows and column entries represent the ECD inputs that transit to the next states of the ECD_RTS-NFA matching machine.

Table 1: ECD 0 cross product of itself and those of ECD (1, 2 and 3).

State	ECD (0x0)	ECD (0x1)	ECD (0x2)	ECD (0x3)
0	0,1,2	0,1,2,3	0,1,2	0,1,2
1	1,2	3	-	-
2	-	-	-	-
3	-	-	-	-
4	-	-	-	-

Table 2: Table of the 2-byte state vectors of four columns merged to form 6 new ECD columns of inputs

State	ECD 0	ECD 1	ECD 2	ECD 3	ECD 4	ECD 5
0	0,1,2	0,1,2,3	0,1,2	0,1,2,3	0,1,2,4	0,1,2,4
1	1,2	3	-	-	4	-
2	-	-	-	-	-	-
3	-	-	-	-	-	-
4	-	-	-	-	-	-

Table 3: 4-byte table of compressed ECDs

	ECD 0	ECD 1	ECD 2	ECD 3	ECD 4
ECD 0	0	1	2	3	4
ECD 1	0	3	2	3	4
ECD 2	2	3	2	3	5
ECD 3	2	3	2	3	5
ECD 4	2	3	2	3	5

B. The Unique Table Synthesis Compression Process

The software parser described in [31] creates blocks of 4-byte matching circuits used for matching regexps at once [32]. Furthermore, *4*-bytes of character inputs are fetched from the BRAMs in Figure 2. The ECD inputs are then used to look-up the equivalent ECDs fetched from the BRAM blocks. The newer compression technique used in the design ensures that the generated ECDs do not grow beyond 128 in number. As such, the process ensures that only 7-bits are needed for compressing each streaming character. This explains why 32-bits enter the BRAM blocks, but only 28-bit value equivalent of the ECD inputs is required as seen in Figure 2. The table of ECDs are then translated into pure logic circuits. The translation module releases an output of compressed <128 bit vector, with each bit position in the vector representing each of the matched 128 ECDs fetched from the BRAM block as seen in Figure 1. The module ensures that the uniquely synthesized table consumes minimal LUTs and other required logic circuits such as Flip-Flops, Multiplexers etc.

Algorithm 2: Hardware synthesis process for the compressed *n*-byte ECDs [32].

```
INPUT: An $k$ x $k$ table of $n$-byte ECD inputs and a 28-bit input from the 2x36kBRAM block described in,
where $n$ = 2 and 4, and $k > 1$.
OUTPUT: A <128-bit vector of compressed ECDs.
BEGIN
    i.   Read the 28-bit inputs from the 2x36kBRAM and the $k$ x $k$ tables of $n$-byte ECDs.
    ii.  Create the relevant 2-dimensional arrays converted into signal variables and initialize the same to
         contain the associated 2-byte and 4-byte tables of compressed ECDs.
    iii. Compute and process the sub-linear table-look up operations to generate the relevant < 128-bit vector
         of outputs. Each bit position of the output bit vector represents an equivalent ECD value.
    iv.  Initialize the tables of 2-byte and 4-byte tables of compressed ECDs. Assign the 1-bit value of '1' to
         the output variable.
END.
```

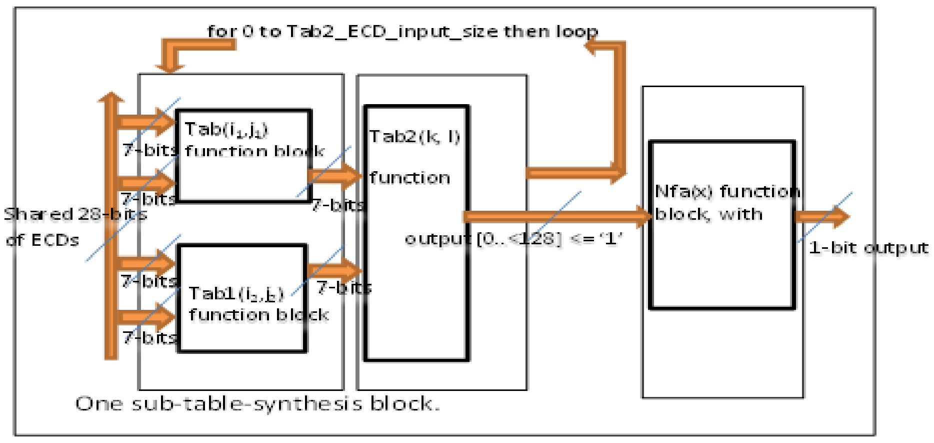
for 0 to Tab2_ECD_input_size then loop
Shared 28-bits of ECDs
7-bits
7-bits
7-bits
7-bits
Tab(i1,j1) function block
Tab1(i2,j2) function block
7-bits
7-bits
Tab2(k, l) function
output [0..<128] <= '1'
Nfa(x) function block with
1-bit output
One sub-table-synthesis block.

Figure 1: The sub-NFA block.

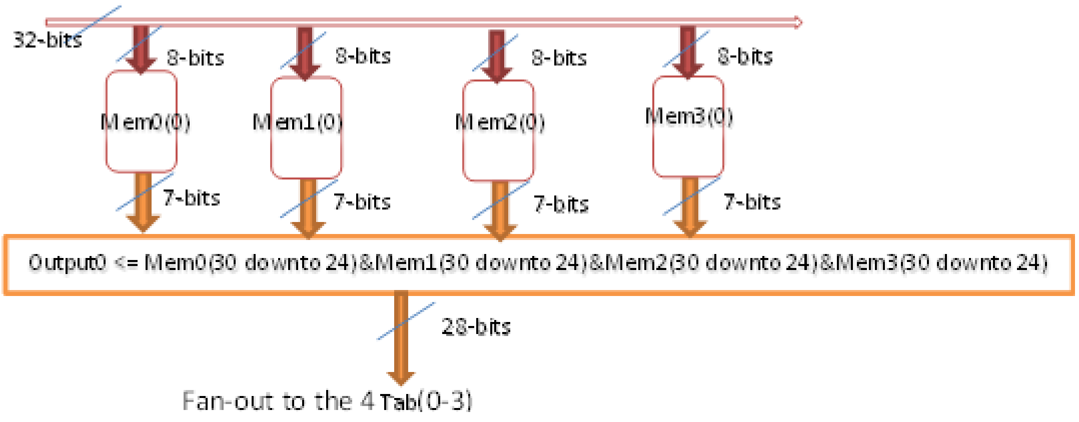
32-bits
8-bits
8-bits
8-bits
8-bits
Mem0(0)
Mem1(0)
Mem2(0)
Mem3(0)
7-bits
7-bits
7-bits
7-bits
Output0 <= Mem0(30 downto 24)&Mem1(30 downto 24)&Mem2(30 downto 24)&Mem3(30 downto 24)
28-bits
Fan-out to the 4 Tab(0-3)

Figure 2: Four 256x8-bit table of ECDs for Mem(0-3)(0) memory blocks,

C. Evaluation

The ECD$_R$TS-NFA design was also implemented using the Xilinx Vixtex-6 device synthesis tool [31], [32]. The design requires only O(n) storage space for the ECDs and O(n^2m) time to process each of the ECDs extracted from each given regexp. It also takes only O(nm) time to search through n patterns with text of length m. A data bus width of 7-bits is used to compute the throughput measured in Gigabits per second (Gbps). As stated earlier on in [31] and [32], creating a design that generates high throughput while incurring minimal logic circuit remains a challenge to many similar pattern matching designs. The formula for computing the design throughput is the same as the one used in [31] and [32]. The compared design approaches are represented by the column heading *design* in Table 4. The *clock speed* (MHz) is the maximum clock frequency attained by each design. The *throughput*, which is the rate at which 4-byte characters are matched per clock cycle, is measured in Gigabits per second (Gbps). The data bus width is 32-bit wide, and the product of the clock rate (MHz) and the data bus width (32-bits) divided by 1024 bits produces the throughput of matching in Gbps. It is the rate at which some workload is achieved.

IV. Discussion

Looking at the various designs in Table 1, it can be observed that the compared approaches are 4-byte character matching designs. Each of result graphs in this section represent the relationship between the various design clock speeds and their respective design throughput. Figure 3 shows how the various designs compare against each other's throughput (Gbps) of matching. The figure shows that the ECD$_R$TS-NFA design throughput is about 3.35% higher than the next highest throughput value reported in [20]. The design throughput is also about 54.54% higher than the least reported throughput value reported by [37]. Also, Figure 4 show that the clock speed of the ECD$_R$TS-NFA design is about 6.21% better than the next highest clock speed reported in [20]. It is also about 54.57% better than the least reported Clock Speed reported by [37]. This shows that the ECD$_R$TS-NFA design holds some promise especially for the future of Intrusion Detection Systems. The results from Table 1 were used to generate the two graphs as seen in Figures 3 and 4.

.

i. Table of Results

Table 1: Table of Design Results [7].

Design Approach	Input	MHz	Throughput
ECD$_R$TS-NFA	4	385.78	12.34
Brodie, Taylor and Cytron [26]	4	133.00	4.26
Sourdis and Pnevmatikatos [36]	4	303.00	9.71
Yamagaki, Sidhu and Kamiya [37]	4	113.40	3.63
Sutton [38]	4	317.19	10.15
Clark and Schimmel [39]	4	218.90	7.00
Yang, Jiang and Prasanna [7]	4	233.13	7.46
Yang and Prasanna [25]	4	300.00	9.60
Yang and Prasanna [41]a.	4	198.6	6.36
Yang and Prasanna [41]b.	4	166.7	5.33
Ganegedara, Yang and Prasanna [40]	4	202.90	6.50
Singapura et al. [20]	4	340.63	11.54

ii. **Chart for the Throughput of matching**

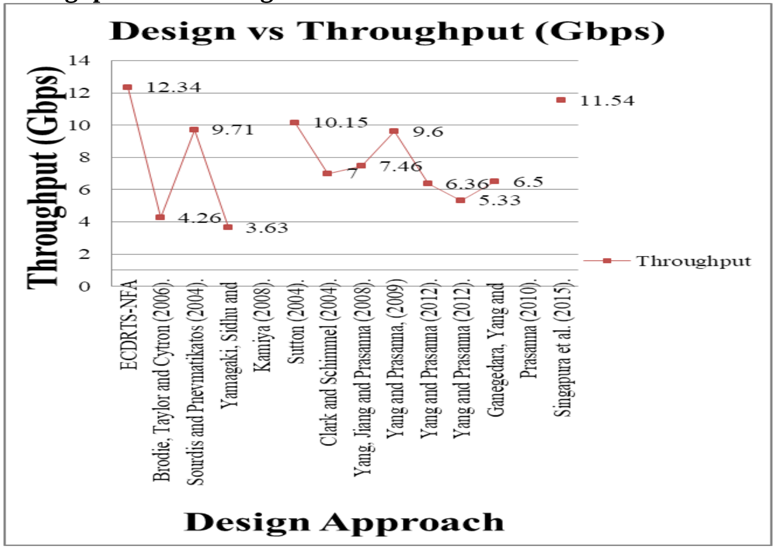

Figure 3: Various designs against their clock speeds (MHz).

iii. **Chart for the Clock Speeds**

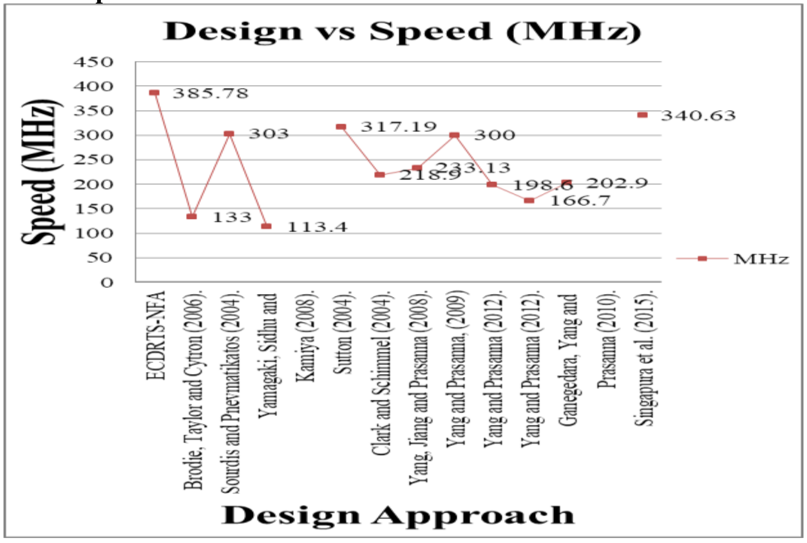

Figure 4: Various designs against their throughput (Gbps).

V. Conclusion

The result as indicated in Figures 3 and 4 show some promise for the ECD$_R$TS-NFA approach. The earlier challenge regarding the amount of time taken to synthesize and placed and routed (PAR) in the ECD-NFA design [32] has been addressed by the ECD$_R$TS-NFA approach. Furthermore, research is still on-going to extend this work, by creating a multi-byte quadruple parallel matching engine version of the ECD$_R$TS-NFA design. The proposed quadruple engine will be able to contain and convert more complex regular expression patterns in such a way that it can fully take advantage of the parallelism provided by FPGAs. The ECD$_R$TS-NFA design was written and implemented using the Java programming language for the software parsing process. It was then fully synthesized and PAR on a Xilinx FPGA Virtex-6 device synthesis tool for the hardware phase. This s the same process described in the design process for the ECD-NFA [31].

For the future work, there is work in progress to scale-up the number of the proposed matching engines to about 10, with each sub-Engine containing 4 parallel engines. Furthermore, with the already optimized memory arrangement and design described in [32], the scaling process for the ECD$_R$TS-NFA should take advantage of the improved memory utilization. It is also hoped that, the scaling process will also improve the Throughput Efficiency (TE) of the design, which is another factor that is continuously been considered in such areas of research. The TE is used to determine the amount of logic resources consumed by related designs, besides just trying to the improve clock speed and throughput of matching.

Acknowledgements
I once more acknowledge the contributions and support of Gerald Tripp towards the success of this continued research work. I also acknowledge the use of the Xilinx ISE Proprietary Project Navigator application version P.49d, volume 14.4 (nt64) which is bundled with the Xilinx Synthesis Tool (XST).

References

[1]. J. Aycock, *Computer Viruses and Malware.* USA: Springer, 2006.

[2]. P. Piyachon and Y. Luo, "Compact state machines for high performance pattern matching, in *Proceedings of the 44th Annual Conference on Design Automation - DAC '07,* 2007, pp. 493-496.

[3]. M. Roesch, Snort - lightweight intrusion detection for networks, in *Proceedings of the 13th USENIX Conference on System Administration, LISA'99,* 1999, pp. 229-238.

[4]. (24 July 2013). *Bro.* Available: http://www.bro.org/download/index.html.

[5]. (24 July 2013). *IOS Intrusion Prevention System Deployment Guide.* [[Cisco IOS Intrusion Prevention System (IPS)] - Cisco Systems]. Available: http://www.cisco.com/en/US/prod/collateral/iosswrel/ps6537/ps6586/ps6634/prod_white_paper0900aecd8062acfb.html.

[6]. M. Becchi and P. Crowley, A hybrid finite automaton for practical deep packet inspection. in *Proceedings of the 2007 ACM CoNEXT Conference on - CoNEXT '07,* 2007b, pp. 1.

[7]. Y. E. Yang, W. Jiang and V. K. Prasanna, "Compact architecture for high-throughput regular expression matching on FPGA." in *Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems - ANCS '08,* 2008, pp. 30-39.

[8]. T. R. Bednar, P. H. Buffet, R. J. Darden, S. W. Gould and P. S. Zuchowski, "Issues and strategies for the physical design of system-on-a-chip ASICs " *IBM Journal of Research and Development,* vol. 46, pp. 661-674, 2002.

[9]. G. Vasiliadis, S. Antonatos, M. Polychronakis, E. P. Markatos and S. Ioannidis, "Gnort: High Performance Network Intrusion Detection Using Graphics Processors." *Recent Advances in Intrusion Detection,* vol. 5230, pp. 116-134, 2008.

[10]. G. Vasiliadis, M. Polychronakis, S. Antonatos, E. P. Markatos and S. Ioannidis, "Regular Expression Matching on Graphics Hardware for Intrusion Detection." *Recent Advances in Intrusion Detection,* vol. 5758, pp. 265-283, 2009.

[11]. C. R. Clark and D. E. Schimmel, "Scalable pattern matching for high speed networks." in *12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines,* 2004, pp. 249-257.

[12]. H. Wang, S. Pu, G. Knezek and J. Liu, "A modular NFA architecture for regular expression matching." in *Proceedings of the 18th Annual ACM/SIGDA International Symposium on Field Programmable Gate Arrays - FPGA '10,* 2010, pp. 209-218.

[13]. W. Jiang and V. K. Prasanna, "Field-split parallel architecture for high performance multi-match packet classification using FPGAs." in *Proceedings of the Twenty-First Annual Symposium on Parallelism in Algorithms and Architectures - SPAA '09,* 2009, pp. 188-196.

[14]. T. T. Hieu, T. N. Thinh, T. H. Vu and S. Tomiyama, "Optimization of regular expression processing circuits for NIDS on FPGA." in *2011 Second International Conference on Networking and Computing.* 2011, pp. 105-112.

[15]. R. Sidhu and V. K. Prasanna, "Fast regular expression matching using FPGAs." in Proceeding FCCM '01 Proceedings of the the 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 2001, pp. 227-238.

[16]. J. E. Hopcroft, R. Motwani and J. D. Ullman, "*Introduction to Automa Theory, Languages and Computation*". Boston, USA: Addison-Wesley, 2001.

[17]. R. Wain, I. Bush, M. Guest, M. Deegan, I. Kozin and C. Kitchen, "An overview of FPGA programming." vol. 1, pp. 4-5, 2006.

[18]. B. L. Hutchings, R. Franklin and D. Carver, "Assisting network intrusion detection with reconfigurable hardware." in *Proceedings. 10th Annual IEEE Symposium on Field-Programmable Custom Computing Machines.* 2002, pp. 111-120.

[19]. P. Bellows and B. L. Hutchings, "JHDL-an HDL for reconfigurable systems." in *Proceeding FCCM '98 Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines,* 1998, pp. 175-184.

[20]. S. G. Singapura et al., "FPGA Based Accelerator for Pattern Matching in YARA Framework". Technical Report. Computer Engineering Technical Report. Los Angeles, CA, USA: Computer Engineering, Ming Hsieh Department of Electrical Engineering-Systems. Report number: CENG-2015-05, 2015, pp. 1-10.

[21]. L. D. Perry, "*Programming by Example*". New York City: McGraw-Hill, 2002.

[22]. G. Tripp, "A Parallel "String Matching Engine" for use in High Speed Network Intrusion Detection Systems ", *Journal in Computer Virology,* vol. 2, pp. 21-34, 2006.

[23]. C. Lin, Y. Tai and S. Chang, "Optimization of pattern matching algorithm for memory based architecture." in *Proceedings of the 3rd ACM/IEEE Symposium on Architecture for Networking and Communications Systems - ANCS '07,* 2007, pp. 11-16.
[24]. A. Mitra, W. Najjar and L. Bhuyan, "Compiling PCRE to FPGA for accelerating SNORT IDS." in *Proceedings of the 3rd ACM/IEEE Symposium on Architecture for Networking and Communications Systems - ANCS '07,* 2007, pp. 127-136.
[25]. Y. E. Yang and V. K. Prasanna, "Software Toolchain for Large-Scale RE-NFA Construction on FPGA " *International Journal of Reconfigurable Computing,* vol. 2009, pp. 1-10, 2009.
[26]. B. C. Brodie, D. E. Taylor and R. K. Cytron, "A Scalable Architecture For High-Throughput Regular-Expression Pattern Matching." *ACM SIGARCH Computer Architecture News,* vol. 34, pp. 191-202, 2006.
[27]. P. Gupta and N. McKeown, "Packet classification on multiple fields." in Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication - SIGCOMM '99. 1999, pp. 147-160.
[28]. J. Bispo, I. Sourdis, J. M.P.Cardoso and S. Vassiliadis, "Regular expression matching for reconfigurable packet inspection." in *2006 IEEE International Conference on Field Programmable Technology,* 2006, pp. 119-126.
[29]. J. T. Arnold, "Lecture Notes for MATH 3034." pp. 1-13, 2007.
[30]. (18 July 2013). Snort IDS/Rules. Available: http://www.snort.org/.
[31]. B. Modi and G.Tripp, "A Highly Compressible Regular Expression Matching Circuit for Network Intrusion Detection Systems". *IOSR Journa of VLSI and Signal Processing (IOSR-JVSP),* vol. 6, Issue 4, ver. II, Jul –Aug, 2016, pp. 50-58, e-ISSN. 2319 -4200, p-ISSN No: 2319 – 4197.
[32]. B. Modi and G.Tripp, "A Synthesizable Memory Grid using BRAMs and Function Generators to Enhance Throughput Efficiency in Regular Expression Pattern Circuits."*IOSR Journa of VLSI and Signal Processing (IOSR-JVSP),* vol. 6, Issue 4, ver. II, Jul –Aug, 2016, pp. 50-58, e-ISSN. 2319 -4200, p-ISSN No: 2319 – 4197.
[33]. R. Rasool, M. Najam, and H.F. Ahmad., et al.,"A novel JSON based regular expression language for pattern matching in the Internet of Things." *Journal of Ambient and Humanized Computing, 25 May, 2018*, pp. 1-19. Springer Berlin. https://doi.org/10.1007/s12652-018- 0869-1.
[34]. K.. Sandeep, V. Srinivasan, and M. Bakthavatchalam, "Real-time regular expression search engine."U.S. Patent 9,967,272, issued May 8, 2018.
[35]. T. Qiu, X. Yang, and B. Wang, "Filtering Techniques for Regular Expression Matching."In: Liu C., Zou L., Li J. (eds) Database Systems for Advanced Applications. DASFAA 2018. Lecture Notes in Computer Science, vol 10829. Springer, Cham.
[36]. I. Sourdis and D. Pnevmatikatos, "Pre-Decoded CAMs for Efficient and High-Speed NIDS Pattern Matching."*Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, Napa, California: IEEE, 2004, pp. 258-267. DOI: 0-7695-2230-0.
[37]. N. Yamagaki, R, Sidhu and S, Kamiya, "High-Speed Regular Expression Matching Engine using Multi-Character NFA."*Proceedings of the 2008 International Conference on Field Programmable Logic and Applications*, Heidelberg, Germany: IEEE, 2008, pp. 131-136. DOI:10.1109/FPL.2008.4629920.
[38]. P. Sutton, "Partial Character Decoding for Improved Regular Expression Matching in FPGAs."*Proceedings of the 2004 IEEE International Conference on Field- Programmable Technology (IEEE Cat. no.04EX921)*. Brisbane, Australia: IEEE, 2004, pp. 25-32. DOI: 10.1109/FPT.2004.1393247.
[39]. C. R. Clark. and D. E. Schimmel, "Scalable Pattern Matching for High Speed Networks."*Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*. Napa, CA, USA: IEEE, 2004, pp. 249-257. DOI: 10.1109/FCCM.2004.50.
[40]. T. Ganegedara, Y. E. Yang and V. K.. Prasanna, "Automation Framework for Large-Scale Regular Expression Matching on FPGA."*Proceedings of the 2010 International Conference on Field Programmable Logic and Applications.* Milano, Italy: IEEE, 2010, pp. 50-55. DOI: 10.1109/FPL.2010.21.
[41]. Y. Yang and V. Prasanna, "High-Performance and Compact Architecture for Regular Expression Matching on FPGA." *IEEE Transactions on Computers*, 61(7), 2012, 1013-1025. DOI:10.1109/TC.2011.129.

Bala Modi1. “An Optimized Multi-Character Pattern Matching Circuit for Network Intrusion Detection Systems.” IOSR Journal of VLSI and Signal Processing (IOSR-JVSP) , vol. 8, no. 7, 2018, pp. 07-14