IOSR Journal of VLSI and Signal Processing (IOSR-JVSP)
Volume 6, Issue 4, Ver. II (Jul. - Aug. 2016), PP 71-77
e-ISSN: 2319 – 4200, p-ISSN No. : 2319 – 4197
www.iosrjournals.org

A Synthesizable Memory Grid using BRAMs and Function Generators to Enhance Throughput Efficiency in Regular Expression Pattern Matching Circuits

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)*

Abstract: *Attacks on various computer networks are usually in form of patterns of attack. The patterns are recognizable mostly based on the data that the respective packet payloads contain. Attack patterns usually occur as strings or regular expression patterns, which are then converted into their equivalent automata. To create an efficient automata, there is a need for the automata design to consume less memory resources per each state of the automata. This is important whenever the design attempts to detect variations of these patterns. This paper explains the design, structure, and suitability of the hardware memory architecture for a Field Programmable Gate Array (FPGA) based automata design. The new design implementation is based on the input compression technique called Equivalence Classification (EC) which is used to drive a Nondeterministic Fine Automata (NFA) referred to as Equivalence Class Decoding NFA (ECD-NFA). The ECD-NFA approach creates classes of compressed inputs represented by positive integers simply called ECDs, which are the class descriptors. The compressed ECDs are then used to drive the automata, instead of unclassified raw character-inputs. This paper further extends the design by creating a memory grid that utilizes half the total number of required primitive block RAMs (BRAMS). Also, by re-writing the algorithm for the table look-up operation, the design utilizes a minimal number of function generators. Function generators are implemented as 6-input look-up tables (LUTs) on the target Xilinx FPGA Virtex-6 device. The efficiency of such LUT-based designs is determined by the throughput efficiency. The throughput efficiency (TE) is computed based on the ratio of LUTs utilized by the states of the automata and the design throughput. The preliminary results obtained for the TE is 3.55, while the clock rate obtained was 440.51MHz*

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

I. Introduction

Most network security systems usually rely on *layers* of *protection* and contain network monitoring and security software. However, as the demand for increased network bandwidths continues to increase, so have the frequency of network attacks and illegal accesses. As such, the consequences to the privacy and confidentiality of both network clients and confidential documents is severe. Attacks appearing in form of *spam, bugs, Denial-of-Service (DoS),* and *malicious software* such as: *viruses* and *worms* etc. have a devastating effect on an entire network system [1]. To determine the patterns in which these attacks occur, pattern matching is required. Pattern matching can either be exact string matching or regular expression matching. Exact string matching belonging to a given packet payload can be performed during the process of deep packet inspection of the packet payload flowing into a given network. The problem with string matching is that it has become inadequate in coping with the current network security challenges. As such, regular expression pattern matching is favoured to deal with the problem attributed to string matching.

Most current software tools such as Snort [2] use *regular expressions* or simply 'regexps' for describing packet payload patterns that are streaming through the network. A regexp is a regular language constructed with character classes defined over a fixed alphabet. A regular language has three basic operations performed on its character classes [3] namely: *concatenation* (.), *union* (|) and *Kleene closure* (*). By properly arranging the three basic operators mentioned, more complex regexps can be constructed. Also, common operators such as: *optionality* (?) and *quantified repetitions* like ({*a*, },{, b} and {*a*, b}), could be constructed by using combinations of the three mentioned basic operators. The hardware exploited in this paper is the Field-programmable Gate Array (FPGA) [4], [6], [11], and [7].

A number of FPGA-based automata implementations exist, which involve the conversion of regular expressions into their respective automata. In [3] it was observed that 'any given pattern that can be matched by a regexp can also be matched by an automaton'. As such, the preferred choice of automata in this paper is the Nondeterministic Finite Automata (NFA). An NFA processing time is capable of being reduced to O(1), but requiring $O(n^2)$ memory [8] on FPGAs. This is only possible by exploiting its fine-grained *parallelism*. Parallelism is considered to be a great advantage that FPGAs have over microprocessors when matching regexps. As a result, NFA-based approaches have gained popularity because of their ability to exploit the parallelism and the *re-configurable* feature available in current FPGAs. The ECD-NFA design described in [5] and [9] classifies all input strings that have the same effect on the automata into classes. The ECDs that are generated are assigned in increasing order to represent the various respective vectors of next states on the transition table, before using them to drive the automata.

However, the challenge has always been with how to reduce the overall input table size, which usually requires a lot of memory. The matching process of the ECD-NFA described in [5] and [9] utilizes primitive Block RAMs (BRAMs). The BRAMs are used for compressing the inputs from raw data inputs to the various equivalence classes as described in Algorithm 1. The equivalence classes are then mapped to their respective ECDs (class descriptors) accordingly. Each class of ECDs represents a set of vector of next states of the NFA. The BRAMs are later synthesized as a grid on a target Xilinx FPGA Virtex-6 device, during the pre-synthesis stage of the design hardware phase as shown in Figure 1. The contents of the BRAMs are then fetched and used to compare against the input strings that are streamed live into the ECD-NFA for a match to occur within the associated NFA engines.

The problem with the current ECD-NFA design is that it utilizes double the number of BRAMs required, and that is not efficient. As such, we shall describe and evaluate the BRAM framework and the *throughput efficiency* of the ECD-NFA design, which is capable of improving the design. This is achieved by comparing the preliminary results to the other known NFA-based designs. The equations for the throughput and throughput efficiency as seen in Equations (1) and (2) are used to determine the efficiency of the related approaches. The ability to utilize minimal logic resources available on the target FPGA, while pursuing higher throughput is a great consideration. The main contribution of this paper is that by creating a memory grid structure, only half of the current memory requirement of the ECD-NFA will be required. Another contribution is that we were able to synthesize the table-look up operations into a small piece of logic, which utilizes minimal LUTs.

The remainder of this paper is organised thus: A brief summary of related approaches is described in Section II. Section III describes the framework and algorithm for BRAM memory grid framework as shown in Figure 1. Section IV discusses the charts of the related designs under consideration based on the results as shown in Figure 4 and Figure 5, and Table 1. The preliminary results obtained for the various related designs are based on the use of the FPGA implementation tools particularly the Xilinx Synthesis Tool (XST) which is bundled with the Xilinx ISE Proprietary Project Navigator application version P.49d, vol. 14.4 (nt64) software package. The obtained results are then plotted and analysed in order to provide a quick and concise view of how each design performs in comparison to the others. Lastly Section V discusses the conclusion and ideas for future work.

II. Related Works

While implementing NFAs as logic, it was observed that if all the source input Flip-Flops (FFs) [10] to the destination input FFs are on ϵ–transitions (epsilon transition), then the FFs can be eliminated [8]. This means that epsilon transitions need not be implemented at all, thereby saving more memory logic circuits. The design described in [8] has pioneered several other approaches that consider building reconfigurable [11] FPGA-based designs. An FSM design was proposed in [12], which operates on a single byte wide data input, while providing a separate FSM for each byte wide data path from a multi-byte input data word. By combining the outputs from the separate FSMs in a given way, string matching is performed in parallel across multiple FSMs per clock cycle. Furthermore, [13] addressed the issue of memory by creating a packed array, having each entry containing the base address of the state vector in the packed array. The address is used for deciding the next states to be visited on the FSM, leading to some significant memory reduction.

The design by [14] memorizes the path that the trigger signals based on specific constraints suitable for both *exact* string matching and complex regexp matching. The design by [15] used a wider input bus through an SRAM interface, which helped to increase the overall throughput of the regular expression matching (REM) block. A heuristic that automatically marshalled *k*-REMs with total *N*-states into *p*-pipelines was implemented by [3]. The process ensured that a function is called to compare every character class within each REM with those previously collected in the BRAM [7]. This happens whenever a REM is to be added to an existing pipeline. The matching outputs of each of the REMs are prioritized, with the higher priority given to the lower-indexed pipelines and stages. The ECD-NFA described in [5] and [9] is constructed into blocks of MCMs, aimed at implementing a classification based approach that is only applicable and appropriate for NFAs. The design is then staged and pipelined in a simple corresponding formation, and utilizes only BRAMs and LUTs.

III. The Memory Grid Framework

The memory grid module described in this section is created from a primitive 36K BRAM to form a block of memory to be used by a single matching regular expression matching (REM) unit as shown in Figure 3. There are two primitive 36K BRAMs that are utilized, and by cleverly splitting each into half using a simple procedure, we obtained 2 x 18K BRAMs from each 36K BRAM. This made it a total of 4 x 18K BRAMs. The 4 x 18K BRAMs were then copied into a grid of 16 x 18K BRAMs thereby forming a virtual memory grid from only 4 x 18K BRAMs. By so doing, up to 4 characters can be matched at the same time by the ECD-NFA. This also translates to a faster matching process as against a single character matching process by a scale of 4:1. Algorithm 1 describes the process of supplying 32-bit input to each of the 4 x 18K BRAM blocks. The topmost bit of each of the 8-bit inputs is shed off, since the maximum number of expected ECDs is 128. This reduces the inputs to a mere 7-bits, which are then concatenated to form a 28-bit wide output from each 4 x 18K BRAM blocks. This memory grid arrangement makes it possible to match up to 4 characters at once. Furthermore, each block of 4 x 18K BRAM can now supply its 28-bit output to at least 4 table decoding blocks as shown in Figure 1. This also means that up to 4 NFA matching units can be supplied with 128 ECDs each from the output of each decoding block. The REM block shown in Figure 1 represents one REM matching block

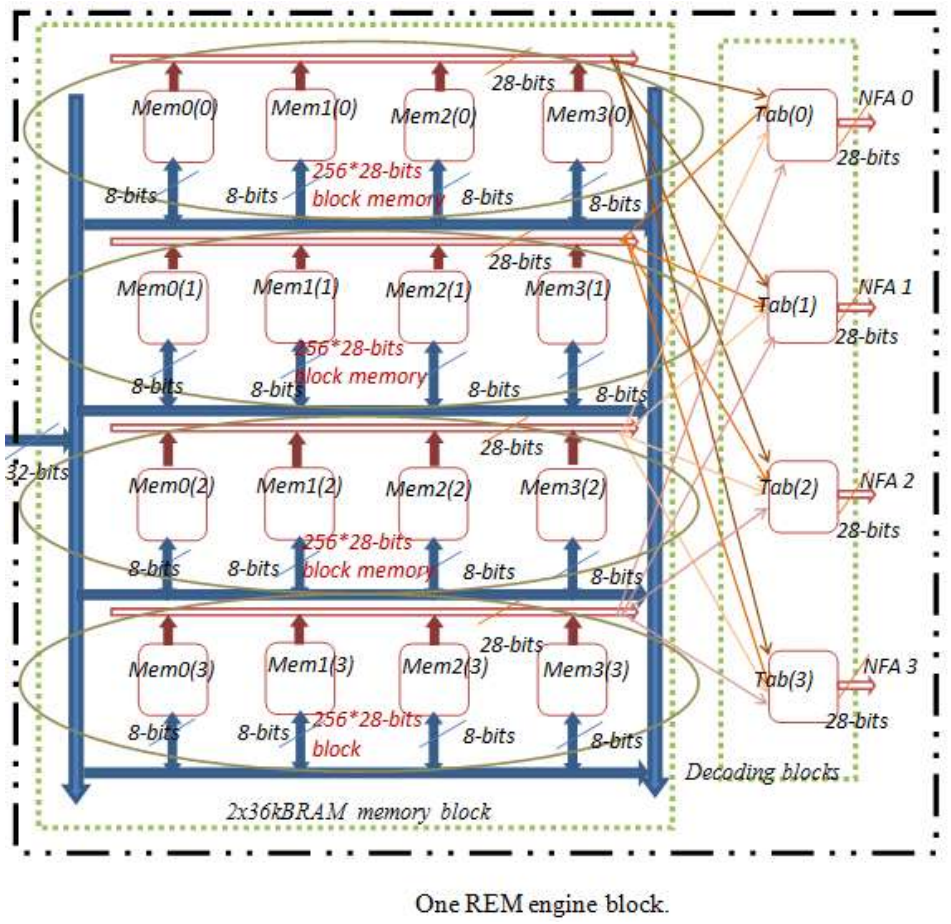


Figure 1: A Memory Grid BRAM Structure [9].

Algorithm 1: Construction of an *n x n* BRAM grid [9].

INPUT: 32-bit input.
OUTPUT: A *<28 bit* output from each REM as shown in Figure 3.
BEGIN

i. *Read and parse the regexps to be constructed into the equivalent ECD-NFAs* and *ECDs.*
ii. *Assign to each classified inputs created in (i) class descriptors (ECDs). The ECD represents the sets of vectors of next states generated during the computation process for all character classes that trigger transitions from a state s_i to a state s_j, where $i < n$, $j < n$.*
iii. *Repeat steps (ii) for ∀ vectors of next states $s_{i,j}$ $i < n$, $j < n$, and store all the sets of vectors of next state in a list of vectors for ∀ states s_i in the ECD-NFA. Compress each raw character input to be streamed by mapping them to their respective ECDs in the BRAM.*
iv. *Finally, exit the process and generate the VHSIC Hardware Description Language (VHDL) code representation of both the 1-byte table of ECDs and the generated ECD-NFA automata. Then automatically upload to the Xilinx FPGA Virtex-6 device for synthesis and implementation.*

END.

A. Synthesizing transition table into LUTs

Once the 4 x 7-bit ECDs are supplied from the 2x36kBRAM (each block of 4 x 18K BRAM) the table synthesis module labelled as 4 x T_c in Figure 3 starts processing. The 4 x 7-bit ECDs are first concatenated into a 28-bit output before it is supplied to the 4 x T_c module. The process then synthesizes the table into a minimal number of LUTs, making it perform table lookup operations faster as described in Algorithm 2. The output of the process is a < 128-bit vector of matching ECDs that are ready to be supplied to the 4 x 7 ECD-NFA automata for matching to take place. Moreover, each of the bit positions in the < 128-bit vector output represents the compressed form of an ECD value. The use of the bit vectors has greatly reduced memory and computation time particularly if it is to be compared to an 8-bit character input representation. The ability to utilize 6-input LUTs for the table look up operation and the compression of the ECDs into bit vectors is a major contribution in this paper.

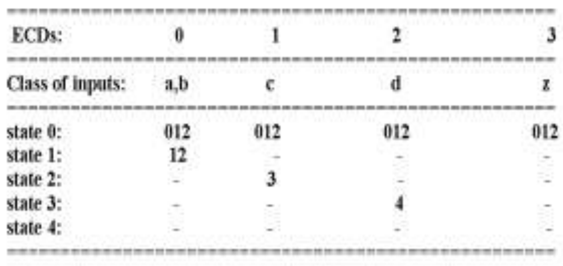
Algorithm 2: Hardware Synthesis Process for the compressed *n*-byte ECDs [9].

INPUT: An *k x k* table of *n*-byte ECD input class descriptors and a 28-bit input from the 4 x 18K BRAM block as shown in Figure 1.
OUTPUT: A *<128-bit* vector of compressed ECDs.
BEGIN

```
i.   Read the 28-bit inputs from the 2 x 36kBRAM 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 1-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 1-byte for the compressed ECDs. Assign the 1-bit value of '1' to the output variable.
```

END.

A sample table of ECDs generated from the regexp "*/(a|b)*(cd)/*" is as shown in Figure 2 as described by [5] and [9]. The various column vectors of next states represent transitions from a given current state to all the various next states to which a given class of ECDs have the same effect on the automata. The top rows of Figure 2 are the various classes of inputs used to represent the sets of vectors of next states on the ECD-NFA, while the columns are the vectors of next states transited to from each given current state to the various next states. Figure 2 shows the various sets of vectors of next states that are transited to for each current state 0-4. The classes of inputs have designated ECDs associated to each one. A description of how the process works has been fully discussed in [5].



ECDs:	0	1	2	3
Class of inputs:	a,b	c	d	z
state 0:	012	012	012	012
state 1:	12	-	-	-
state 2:	-	3	-	-
state 3:	-	-	4	-
state 4:	-	-	-	-

Figure 2: Table of ECDs [5] and [9].

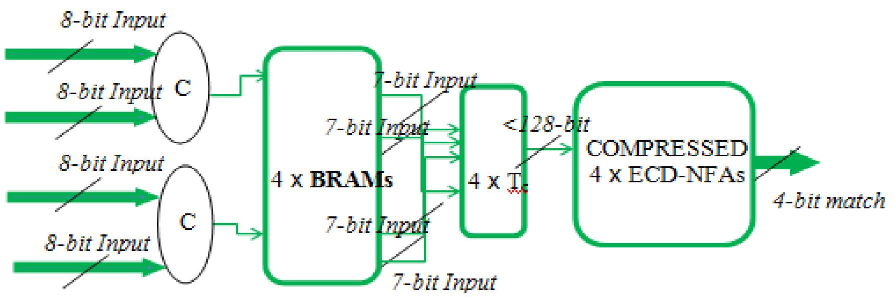


Figure 3: Block diagram of four *1*-byte input ECD-NFA REM matching unit [9].

B. Evaluation

The ECD-NFA design was evaluated using the Xilinx Vixtex-6 architecture. The design required only O(*n*) storage space for the ECDs and O(*n*) time to process each of the ECDs. The ECD-NFA takes O(*nm*) time to search through a text of length *m*. A data bus width of 8-bits was used to compute the throughput measured in Gigabits per second (Gbps). Afterwards, the ratio for the look-up tables (LUTs) utilized by states of each ECD-NFA automata was recorded. Lastly, the clock rate, and throughput, together with the computed LUTs per state ratio was used to compute the throughput efficiency [3] as seen in Table 1. Having a design that obtains high throughput while incurring minimal logic resource cost has always been a trade-off and challenge with such REM-based designs. The Equation (1) and (2) are used to compute the throughput and throughput efficiency.

$$\textit{Throughput} = \textit{clock rate} * \textit{data bus width} \qquad (1)$$

$$\textit{Throughput efficiency} = \frac{\textit{Throughput} * \textit{No. of states}}{\textit{No. of LUTs}} \qquad (2)$$

Table 1: Table of Results [3] and [5].

Design	**Clock Rate**	**Throughput**	**LUTs/State**	**Throughput Efficiency**
Non-ECD-NFA	290.12	2.27	2.10	1.08
ECD-NFA	440.51	3.44	0.97	3.55
Bispo et al. [16]	362.50	2.9	1.28	2.3
Clark and Schimmel [4]	250.00	2.00	1.70	1.2
Mitra, Najjar and Bhuyan [15]	100.78	0.8	2.3	0.35

IV. Discussion

The discussion dwells on how the various designs compare against each other. Figure 4 and Figure 5 give a pictorial view of the results shown in Table 1.

A. Chart

The value of the ECD-NFA's LUTs/state ratio is about 43% lower than the next lowest reported ratio [16] as seen in Figure 4. The throughput efficiency reported against our ECD-NFA design is about 60% higher than the next highest reported throughput efficiency as seen in Figure 5. This is still work in progress, but there is hope that the optimised version that is currently being developed will bring much improvement to the current design's clock rate and throughput.

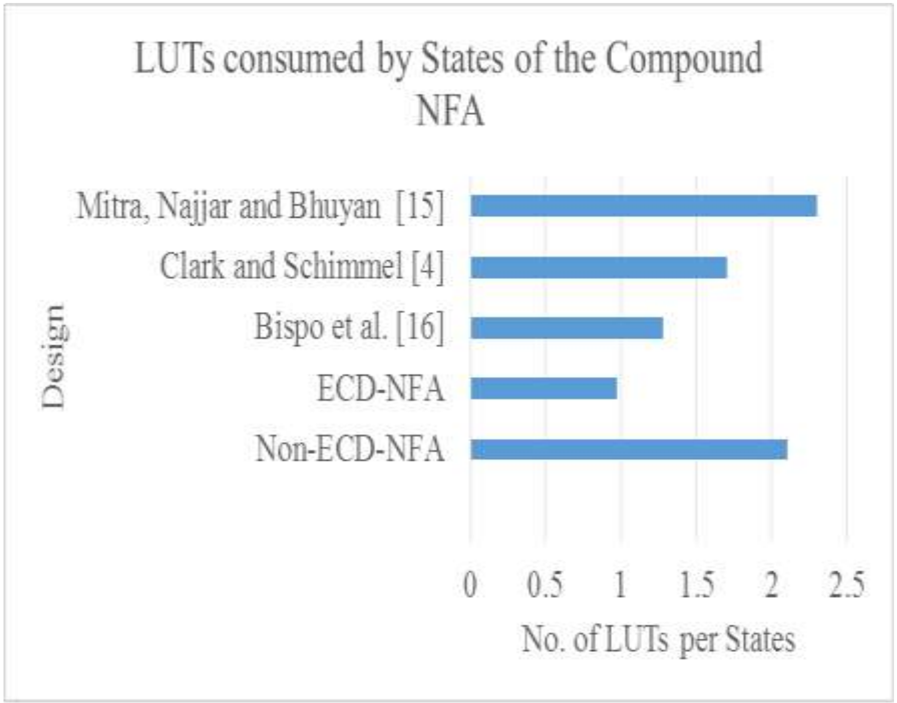


Figure 4: The Number of LUTs consumed by the States of the Compound NFA.

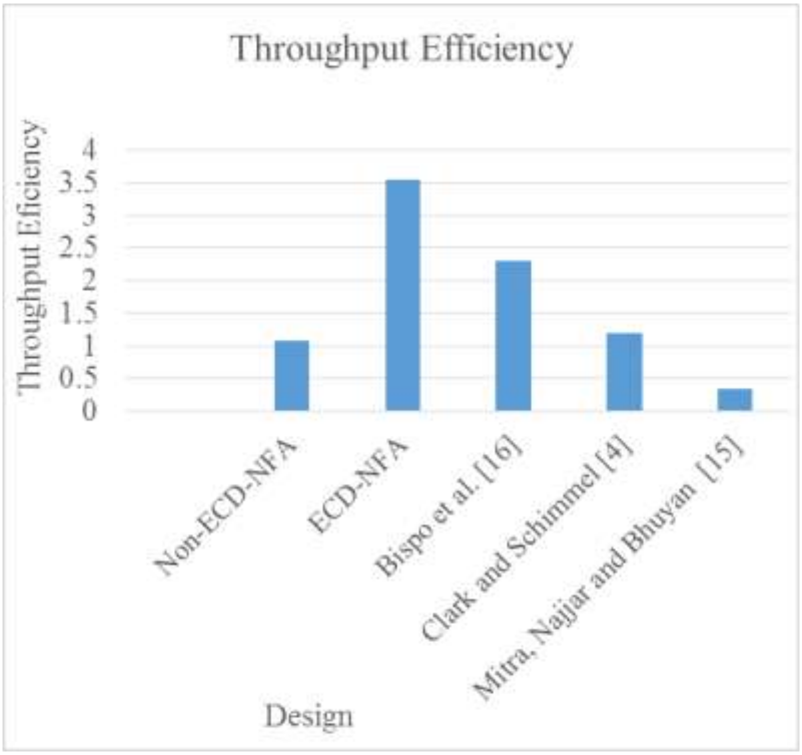


Figure 5: Throughput Efficiency.

V. Conclusion And Future Work

The memory grid framework developed in the ECD-NFA automata holds some promise. Although it is still early to determine, but there is the likelihood that the grid framework will prove useful when considering the construction a parallel multi-character and multi-pattern approach. The approach will serve as the optimized version of the existing ECD-NFA which at the moment is still not very efficient. The lack of efficiency is attributed largely to the large number of LUTs, 128-bit wide shift registers, multiplexers, FFs and other logic circuits that the design utilizes. We are optimistic that the time it takes to synthesize, and PAR the design will be greatly reduced when the optimized ECD-NFA design is fully developed. The new design will be much bigger and capable carrying out scalable parallel multiple-character and multiple-pattern matching of ECDs. We are optimistic that the proposed optimized design will increase the margin of the current ECD-NFA design throughput and throughput efficiency by a scale of 4:1.

Acknowledgements

I acknowledge the tireless and professional contribution of Gerald Tripp for making this design concept a reality. I also acknowledge the effort of Xilinx Incorporation for providing the study pack license to researchers like me for the Xilinx ISE Proprietary Project Navigator application version P.49d, vol. 14.4 (nt64) software package. The software was used for synthesizing and implementing the grid based ECD-NFA design.

References

[1]. J. Aycock, Computer Virus and Malware, N.Y, USA: Springer, 2006.
[2]. Snort, "Snort IDS/Rules," [Online]. Available: http://www.snort.org/. [Accessed 18 July 2013].
[3]. 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 onArchitectures of Networking and Communications Systems-ANCS'08, San Jose, CA, USA, 2008.
[4]. C. Clark and D. Schimmel, "Scalable Pattern Matching for High Speed Networks," in 12th Annual IEEE Symposium on Field-Programmable Custom Computing Matchines, Washington DC, USA, 2004.
[5]. B. Modi and G. Tripp, “A Highly Compressible Regular Expression Matching Circuit for Network Intrusion Detection Systems: An ECD-NFA Approach,” Journal of VLSI and Signal Processing (IOSR-JVSP), vol. 6, no. 4, Ver II, pp. 50-58, 2016.
[6]. Y. E Yang and V. Prasanna, "Software Toolchain for Large-Scale RE-NFA Construction on FPGA," International Journal of Reconfigurable Computing, 2009, no. 2, pp. 1-10, 2009.
[7]. T. Hieu, T. Thinh, T. Vu and S. Tomiyama, "Optimization of Regular Expression Processing Circuits for NIDS on FPGA," in Second International Conference on Networking and Computing, Osaka, Japan, 2011.
[8]. R. Sidhu and V. Prasanna, "Fast Regular Expression Matching using FPGAS," in Proceedings of the 9th Annual IEEE Symposium of Field-Programmable Custom Computing Machines-FCCM'01, Rohnert Park, CA, USA, 2001.

[9]. B. Modi, FPGA-based High Throughput Regular Expression Matching for Network Intrusion Detection Systems, Canterbury. Kent UK, Kent: Thesis, 2015, pp. 1-155.

[10]. M. Trefzer, J. Walker, S. Bale and M. Tyrel, "Fighting Stochastic Variability in a D-type Flip-Flop with Transistor-Level Reconfiguration," IET Computers and Digital Techniques, vol. 9, no. 4, pp. 190-196, 2015.

[11]. 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, Monterey, CA, USA, 2010.

[12]. G. Tripp, "A Parallel String Matching Engine for use in High Speed Network Intrusion Detection Systems," Journal in Computer Virology, 2(1), 21-34, 2006.

[13]. G. Tripp, "Regular Expression Matching with Input Compression and Next State Prediction," Kent Academic Repository, University of Kent Press, Canterbury, 2008.

[14]. 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, Orlando, Florida, USA, 2007.

[15]. A. Mitra, W. Naijjar 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, Orlando, Florida, USA, 2007.

[16]. J. Bispo, I. Sourdis, J. Cardoso and S. Vassiliadis, "Regular Expression Matching for Reconfigurable Packet Inspection," in Proceedings of IEEE International Conference on Field Programmable Technology, Bangkok, Thailand, 2006.