Fast Regular Expression Matching Using FPGA

Jan Kořenek[*]
Department of Computer Systems
Faculty of Information Technologies
Brno University of Technology
Božetěchova 2
612 66 Brno, Czech Republic
korenek@fit.vutbr.cz

Abstract

With the growing number of viruses and network attacks, Intrusion Detection Systems have to match a large set of regular expressions at multi-gigabit speed to detect malicious activities on the network. Many algorithms and architectures have been designed to accelerate pattern matching, but most of them can be used only for strings or a small set of regular expressions. The capacity of available FPGA chips is a limitation for architectures based on a nondeterministic finite automaton. Therefore we propose new algorithm to find a non-collision set of states which enables to map a part of the transition table to the memory instead of the FPGA logic cells. For all analysed sets of regular expressions, the algorithm was able to find a non-collision set with 61.4 % of states in average and a non-collision set with 83.6 % of states for the best case. System of Parallel Automaton Parts is introduced, it is a model which represent a division of the automaton by sets of states. New NFA Split architecture is proposed for mapping of the model to the FPGA. As non-collision sets of states are mapped to the hardware architecture with embedded memory blocks, the amount of consumed flip-flop registers and look-up tables is significantly decreased. For all tested sets of regular expressions, the NFA Split architecture reduces the amount of consumed flip-flops to 43.3 % and look-up tables to 66.8 % in average.

Categories and Subject Descriptors

C.2.0 [**Computer Communication Networks**]: General—*Security and protection (e.g., firewalls)*

Keywords

Regular expression matching, finite automaton, FPGA

[*]Recommended by thesis supervisor:
Prof. Václav Dvořák
Defended at Faculty of Information Technologies, Brno University of Technology on December 2, 2010.

Kořenek, J.: Fast Regular Expression Matching Using FPGA. Information Sciences and Technologies Bulletin of the ACM Slovakia, Vol. 2, No. 2 (2010) 103-111

1. Introduction

The growth of computer networks provides more opportunities for suspicious activities. The amount of worms, viruses and network attacks is steadily increasing. Suspicious activities on a network can be detected by Intrusion Detection Systems (IDS), where the most important operation is pattern matching in packet payload. Pattern matching is a time-critical task and current processors are not able to match patterns on multi-gigabit networks without packet loss, even if they use the best algorithms. IDS systems use hardware acceleration of pattern matching to speed-up processing of network traffic and achieve multi-gigabit throughput.

In recent years, researchers have introduced many hardware architectures [1, 2, 7, 18–21, 23] for pattern matching based on FPGA and ASIC technology. Most of presented approaches are able to work at multi-gigabit speed and support large set of patterns, but can operate only with strings and cannot be easily extended for regular expressions. Vern Paxson has shown [14] that regular expressions are more powerful for intrusion detection. Several papers have presented architectures for regular expression matching [5, 6, 15] with significant hardware acceleration, but only with small set of regular expressions.

As the amount of regular expressions for intrusion detection is steadily increasing, regular expression matching remains a serious challenge. Therefore we propose new NFA Split architecture which reduces the amount of consumed FPGA resources in order to match larger set of regular expressions at multi-gigabit speed. The proposed reduction uses model of nondeterministic (NFA) and deterministic (DFA) finite automaton for effective mapping of regular expressions to FPGA. We have designed an algorithm to find non-collision set of states and split the NFA automaton to deterministic and nondeterministic parts. A System of Parallel Automaton Parts is introduced to represent the division of the automaton. In NFA Split architecture, deterministic parts are mapped to memory units and the nondeterministic part into the FPGA logic. We have observed that significant part of the transition table can be stored in a memory, which yields less FPGA resource consumption and support of more regular expressions.

The paper is divided into the following sections: Section 2 briefly summarises related work for pattern matching, while Section 3 introduces analysis of mapping nondeterministic automata to FPGA. An algorithm to find non-collision set of states is proposed together with system of parallel automaton parts in Section 4. In Section 5 NFA Split architecture and synthesis of regular expressions into hardware matching units is described. Section 6 describes experimental results obtained by evaluation of proposed architecture and algorithms, and finally, Section 7 concludes our work and suggests next possible ways of our research.

2. Related Work

In recent years, many researchers have proposed high-speed pattern matching hardware architectures. Sourdis et al. proposed an architecture based on parallel comparators [19] and pre-decoded CAM [18]. Baker et al. introduced an acceleration of the KMP algorithm [2] and synthesis of patterns to FPGA [1, 23] which uses a tree-based hardware strategy to share FPGA resources among multiple patterns. In [20] Tan et al. propose an efficient algorithm that converts an Aho-Corasick automaton into multiple binary state machines in order to reduce space requirements. Several approaches use an off-chip memory to store a set of patterns and then reduce communication with the memory by hash functions [4] or Bloom filters [7, 8].

Vern Paxson has shown [14] that regular expressions are more powerful for detection of suspicious activities on the network. Many string matching architectures are fast but cannot be extended to regular expressions. Mapping of regular expressions to custom hardware was first explored by Floyd and Ullman [9], who showed that an NFA can be implemented using a programmable logic array. Sindhu et al. [15] proposed efficient mapping of NFAs to FPGA and Clark et al. improved the mapping by shared decoder [5, 6] which significantly reduces amount of consumed logic resources.

Kumar et al. [11, 12] and Yu et al. [22] proposed to use memory-based architectures which use a DFA to represent set of regular expressions. As DFA representations require large amount of memory, algorithms to reduce DFA size were introduced. Yu et al. [22] have proposed an efficient algorithm to partition a large set of regular expressions into multiple groups, such that overall space needed by the automata is reduced dramatically. Kumar analysed influence of regular expressions on the size of DFA [11] and proposed to represent regular expressions by Delayed Input DFA (D2FA) [12]. In [3] Betcchi introduced first architecture which combines NFA and DFA. She use a DFA automaton and converts *dot-star* sub-expressions to NFA automata in order to reduce memory requirements. As the architecture uses a memory to store NFA transition table, the architecture can be easily overwhelmed if an attacker prepare an appropriate network traffic. Despite these optimisations the size of models derived from DFA are large.

We propose new algorithm to find a non-collision set of states which enables to map a part of the transition table to the memory instead of the FPGA logic cells. The algorithm does not analyse regular expressions but combines a model of deterministic and nondeterministic automata. System of Parallel Automaton Parts is introduced as a model which represents a division of the automaton by sets of states. New NFA Split architecture is proposed for mapping of the model to the FPGA in order to reduce the amount of consumed flip-flop registers and look-up tables.

	1 regular expression of length n		m regular expressions of length n together	
	Processing complexity	Storage cost	Processing complexity	Storage cost
NFA	$O(n^2)$	$O(n)$	$O((nm)^2)$	$O(nm)$
DFA	$O(1)$	$O(\Sigma^n)$	$O(1)$	$O(\Sigma^{nm})$

Table 1: Worst case comparisons of DFA and NFA time and space complexity.

3. Analysis of Existing Architectures

For a regular expression matching, deterministic and nondeterministic finite automata are used. The advantage of deterministic automata is a linear time complexity in the worst case. One input symbol is processed in every clock cycle. This means that the matching speed can be guaranted. On the other hand the disadvatage of DFA is a space complexity. Due to the determinstation of the automaton, the amount of states grows exponentially and consequently the size of the transition table grows exponencially too. Then it is a problem to find a large and fast enough memory to store the transition table. NFA has a linear space complexity with respect to the length of the regular expression, but have to cope with nondeterminism. If backtracking is used, one input symbol is processed with $O(n^2)$ time complexity, where n is the number of automaton states. The worst case time complexity to process one input symbol and space requiremets for DFA and NFA is shown in Table 1.

Parallel processing can increase matching speed or decrease memory requirements. A simple acceleration technique of regular expression matching in hardware is to use multiple parallel units, which brings up an overhead with data distribution and with repliacation of data structures. Therefore many approaches use parallel processing at the level of automaton. For example a nondeterministic automaton can use parallel processing to compute multiple next states and reduce backtracking. If k next states can be computed at once, the time complexity to process one input symol is reduced to $O((\frac{nm}{k})^2)$. For the deterministic automaton, parallel processing can be used to reduce memory requirements. The set of regular expressions can be divided into k subsets and every subset can be matched by a different matching unit. The memory requirements are reduced to $O(k \cdot \Sigma^{\frac{m}{k}n})$. Parallel processing can be used also to transform the input alphabet or to compress the transition table.

FPGA technology provides massive parallel processing by look-up tables, flip-flop registers and embedded memory blocks. Massive parallel processing can be used to accelerate regular expression matching with nondeterministic automata. Sidhu and Prasanna [16] introduced the first mapping technique of nondeterministic automaton to the FPGA in order to accelerate regular expression matching. The authors represent every state by one flip-flop register (one bit), which can be set to logical one or zero. According to the stored value the state is active or inactive. As every state can be set independently of each other to logical one or zero, any subset of states can be active concurrently. The architecture solve the nondeterministic choice of next state by activation of all next states which can be reached by executable transitions. Therefore every input symbol can be processed in one clock cycle. Every transition is represented by an AND gate between state and a comparator of the input symbol and a, where $a \in Sigma$ is the symbol that labels the transition. An OR gate is used to combine all signals from AND gates which represent input transitions to given state. Clark [5, 6] has improved the mapping of the NFA to the FPGA by a shared decoder which transforms input symbols to individual signals. Using the shared decoder, transitions can be reresented by a two-input AND gates instead of 8-bit comparators, which significantly reduces amount of consumed look-up tables.

Current approaches for mapping of NFA to the FPGA enables to have active any subset of states. Although more than only one state can be concurrentlly active in an NFA, usually many states are inactive when an input symbol is accepted. Flip-flop registers of inactive states and corresponding next state logic is not used to calculate next states. It means that in every clock cycle only part of the FPGA logic is used.

The ratio between concurrently active and inactive states corresponds to the ratio between used and unused logic to calculate next states of the automaton. The maximum amount of concurrently active states can be determined from relations between nondeterministic and deterministic models of finite automata. For a DFA $A^D = (N^D, \Sigma^D, \delta^D, q_0^D, F^D)$, every state $q^D \in Q^D$ is defined by set of states $q^D \subseteq Q$ from the original NFA $A^N = (Q, \Sigma, \delta, q_0, F)$. Moreover, every set of states, which can be active concurrently in the NFA, is represented by one state in the DFA. Therefore we can find state q^D_{max} which corresponds to the maximum amount of concurrently active states in the NFA:

$$\forall q^D \in Q^D : |q^D_{max}| \geq |q^D| \tag{1}$$

We performed an analysis how effectively the FPGA logic is used in current mapping techniques of NFA to the FPGA. For the analysis, we used regular expressions of L7 decoder [13] and five selected modules of Snort [10]. These sets of regular expressions were transformed to an NFA and then the created automaton was determinised. After that selected a state q^D in the DFA according to Equation 1 which corresponds to the maximum amount of concurrently active states in the NFA. The results of the analysis are summarised in Table 2.

The table contains for all sets of regular expressions the total amount of NFA states in the first row, the maximum amount of concurrently active states in the second row and the proportion between maximum amount of concurrently active states and all NFA states in percent in the third row. We can see that for all selected sets of regular expressions less than 4 % of states can be concurrently active. It means that in every clock cycle less than 4 % of FPGA resources are used to calculate next states and the remaining 96 % are not used.

As every set of states which can be active concurrently in the NFA is represented by one state in the DFA, we can analyse the common size of set of concurently active states in NFA. For regular expression of the L7 decoder, we have created a histogram of DFA states in Figure 1. In the histogram, all DFA states $q^D \in Q^D$ are split to bars according to the $|q^D|$, where q^D is the set of concurrently active states in the NFA.

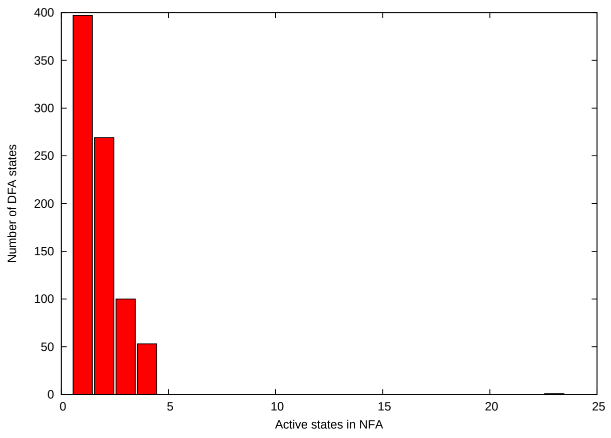

Figure 1: Histogram of DFA states. All DFA states $q^D \in Q^D$ are split to bars according to the $|q^D|$, where q^D is the set of concurrently active states in the NFA.

In Figure 1 we can see that most of DFA states are defined by set with one or two NFA states. It means that only one or two flip-flop register and corresponding next state logic is commonly used to calculate next states in current mapping techniques of NFA to FPGA. Moreover, all sets of concurrently active states contain less than 4 % of all NFA states. Results of the analysis show oportunities for a new more efficient mapping technique of NFA to the FPGA.

4. System of Parallel Automaton Parts

The result of the analysis is that the majority of states are not active and do not need to access transition table at the same time. It can be considered for efficient mapping of NFA to the FPGA and store a part of the transition table in embedded blocks of memories (BlockRAMs). Two concurrently active states can cause a collision in memory access if both states have transitions stored in the same memory. Therefore we call two states which can be concurrently active as *states with collision*. As an opposite to collision states we call two states which cannot be concurrently active as *states without collision* or *non-collision states*.

Definition 1. (States without collision) Let A be an NFA $A = (Q, \Sigma, \delta, q_0, F)$. Two states $q_i, q_j \in Q$, $q_i \neq q_j$ are called *states without collision* or *non-collision states*, if for any input string $w \in \Sigma^*$ does not exist a sequence of configurations:

$$(q_0, w) \vdash^* (q_i, \varepsilon)$$

$$(q_0, w) \vdash^* (q_j, \varepsilon)$$

Similarly we will use these terms for set of states. Transitions for a set of states without collisions (non-collision set of states) can be stored in a memory instead of FPGA logic, because single access to the memory is guaranteed. Set of states without collision can help not only to store a part of the transition table in memory, but also enables to improve state encoding. As states in the set cannot be concurrently active, binary encoding can be used. Then

	L7 dec.	Snort 1	Snort 2	Snort 3	Snort 4	Snort 5
All states of NFA [-]	774	3888	2774	1060	1038	819
Max. active states [-]	23	122	19	18	25	32
Max. active states [%]	2,97	3,14	0,68	1,70	2,41	3,91

Table 2: The maximum amount of active states in nondeterministic automaton for regular expressions of L7 decoder and for regular expressions of five selected modules of Snort.

the current state of the automaton is represented by less bits and consequently less flip-flops are used. We propose an algorithm which has an NFA $A = (Q, T, \delta, q_0, F)$ as the input and calculates the set of states without collision Q^{nca}. The algorithm consists of the following 5 steps:

1. Transform NFA $A = (Q, \Sigma, \delta, q_0, F)$ to DFA $A^D = (Q^D, \Sigma, \delta^D, q_0^D, F^D)$, where $Q^D \subseteq 2^Q$.
2. For all states $q_i \in Q$ create the set $S_{q_i}^{ca}$ which contains collision states with q_i:
   $S_{q_i}^{ca} = \{q_j \in Q \mid q_i \neq q_j \wedge \exists q^D \in Q^D : q_i, q_j \in q^D\}$.
3. Let $Q^{nca} = Q$.
4. Keep removing collision states from the set Q^{nca} until the set contains only states without collisions:
   (a) select a state $q_{max} \in Q^{nca}$ with the largest set of states $S_{q_{max}}^{ca}$:
   $\forall q_i \in Q^{nca} : |S_{q_{max}}^{ca}| \geq |S_{q_i}^{ca}|$
   (b) remove q_{max} from Q^{nca},
   (c) for all states $q_i \in Q^{nca}$ remove q_{max} from the set $S_{q_i}^{ca}$ and
   (d) if $\exists q_i \in Q^{nca} : S_{q_i}^{ca} \neq \emptyset$ then go to (a).
5. Q^{nca} is the set of states without collision.

First, concurrently active states are detected by a transformation of the automaton. Then the Q^{nca} is set to the whole set of NFA states Q and collision states are subsequently removed, until no state in Q^{nca} has a collision. The state to be removed is selected according to the heuristic, which is based on the number of collision states $|S_q^{ca}|$. It means that states with the most collisions are removed first. As the algorithm does not check all subsets of Q, it might not find the optimal solution. On the other hand, the algorithm provides very good results for all inspected sets of regular expressions. In average, more than 61% of states were identified as states without collision Q^{nca}, which means that a significant part of the transition table can be stored in memory.

The proposed algorithm can be applied again to the set $Q^N = (Q \setminus Q^{nca})$ in order to obtain next set of states without collision. Generally, if the algorithm is applied k times, it can find k sets without collision $Q_1^{nca}, ...Q_k^{nca}$. Then every set $Q_i^{nca} \subseteq Q, i \in \langle 1; k \rangle$ determines a part of the NFA $A = (Q, \Sigma, \delta, q_0, F)$ which has the state–transition function restricted to the set of states Q_i^{nca}.

Definition 2. (Part of the automaton determined by a set of states). Let $A = (Q, \Sigma, \delta, q_0, F)$ be an NFA and $Q^s \subseteq Q$ is a set of states. Then the set of states Q^s determines the part of the automaton $A/_{Q^s}$, which is defined by tuple $A/_{Q^s} = (Q^s, Q_{in}, Q_{out}, \Sigma, \delta^s, q_0^s, F^s)$, where

- $Q^s \subseteq Q$ is the set of internal states.
- $Q_{in} = \{q_s \mid q_s \in Q^s \wedge q_s \in \delta(q, a) \wedge q \in (Q \setminus Q^s)\}$ is the set of input states.
- $Q_{out} = \{q \mid q \in (Q \setminus Q^s) \wedge q \in \delta(q_s, a) \wedge q_s \in Q^s\}$ is the set of output states.
- Σ is the input alphabet.
- $\delta^s : Q^s \times \Sigma \to 2^Q$ is the state-transition function restricted to the set of states Q^s. For a state $q_{src} \in Q^s$ a $q_{dst} \in Q$ and an input symbol $a \in \Sigma$ of transition $q_{dst} \in \delta^s(q_{src}, a)$ is defined only if the transition $q_{dst} \in \delta(q_{src}, a)$ is defined.
- q_0^s is the inital state of the automaton part which is defined as:
  $$q_0^s = \begin{cases} q_0 & \text{for} \quad q_0 \in Q^s \\ idle & \text{for} \quad q_0 \notin Q^s \end{cases}$$
- $F^s \subseteq F$ is the set of final states restricted to Q^s:
  $$F^s = F \cap Q^s$$

Definition 3. (Deterministic part of the automaton). Let $A = (Q, \Sigma, \delta, q_0, F)$ be a nondeterministic automaton and $Q^s \subseteq Q$ is a set of states. Part of the automaton $A/_{Q^s} = (Q^s, Q_{in}, Q_{out}, \Sigma, \delta^s, q_0^s, F^s)$ is called *deterministic part of the automaton*, if for any input symbol $a \in \Sigma$ and state $q_s \in Q^s$ exists at most one state $q \in Q$ such that $q \in \delta^s(q_s, a)$. If the part of the automaton is not deterministic, the part is called nondeterministic.

As at most one next state is defined for current state and input symbol in deterministic part of the automaton, we can define state-transition function for deterministic part of the automaton $A/_{Q^s} = (Q^s, Q_{in}, Q_{out}, \Sigma, \delta^s, q_0^s, F^s)$ as:

$$\delta^s : Q^s \times \Sigma \to Q \qquad (2)$$

As two states cannot be active in the set of states without collision, the part of the automaton determined by the set of states without collision is deterministic. Using the proposed algorithm, we can find in the automaton $A = (Q, \Sigma, \delta, q_0, F)$ sets of states without collisions $Q_1^{nca}, ...Q_k^{nca}$ which divide the automaton to $(k+1)$ parts $A/_{Q_1^{nca}}, ...A/_{Q_k^{nca}}$ and $A/_{Q^N}$. The part $A/_{Q^N}$ is determined by the set $Q^N = (Q \setminus \bigcup_{i=1}^{k} Q^{nca})$. We do not know any specific information about the part $A/_{Q^N}$. Nevertheless parts determined by sets of states without collisions $A/_{Q_1^{nca}}, ...A/_{Q_k^{nca}}$ are deterministic, which can be utilized for mapping to the hardware architecture.

Generally, if an automaton is split to multiple parts with specific characteristics, the known characteristics can be utilized during the mapping to the hardware architecture.

Therefore we define for an automaton $A = (Q, \Sigma, \delta, q_0, F)$ *System of Paralel Automaton Parts* $A/_{[Q^1,Q^2,...Q^k]}$, which splits the automaton to k parts determined by sets of states $Q^1, Q^2, ..., Q^k \subseteq Q$.

Definition 4. (System of Paralel Automaton Parts) Let $A = (Q, \Sigma, \delta, q_0, F)$ be an automaton and sets of states $Q^1, Q^2, ..., Q^k \subseteq Q$ determine k different parts of the automaton $A/_{Q^1}, A/_{Q^2}, ...A/_{Q^k}$. *System of Parallel Automaton Parts* $A/_{[Q^1,Q^2,...Q^k]}$ is defined by set of states $Q^1, Q^2, ..., Q^k$, if

$$Q = \bigcup_{i=1}^{k} Q^i$$

For the System of Parallel Automaton Parts, the only one condition is that the union of sets of states $Q^1, Q^2, ..., Q^k$ have to be equal to the set of all automaton states Q. Then transitions of the automaton can be defined between states which belong to different parts of the automaton. If every part is mapped to a single hardware unit, these transitions can be viewed as communication between two different hardware units. Generally, all automaton parts can be connected by transition one to each other. Then all hardware units which represent automaton parts have to be fully interconnected and $\frac{k(k-1)}{2}$ bidirectional lines are needed to connect k hardware units (automaton parts).

The communication model can be significantly simplified if System of Parallel Automaton Parts has a central part and all transitions between two different automaton parts enter or leave the central part. Then the number of bidirectional connection lines is reduced from $\frac{k(k-1)}{2}$ to $(k-1)$. We can see in Figure 2 the difference in communication model between System of Automaton Parts $A/_{[Q^0,Q^1,Q^2,Q^3,Q^4]}$ without a central part (Figure 2a) and with the central part $A/_{Q^0}$ (Figure 2b).

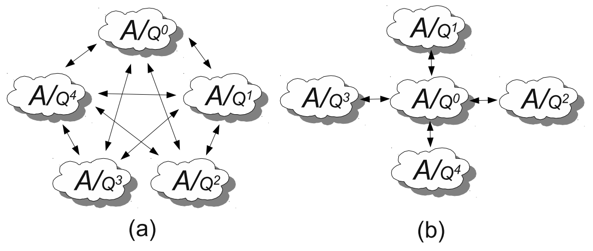

Figure 2: The difference in the communication model between System of Automaton Parts $A/_{[Q^0,Q^1,Q^2,Q^3,Q^4]}$ (a) without a central part and (b) with the central part $A/_{Q^0}$.

Definition 5. (Centralised system of automaton parts) Let $A/_{[Q^1,Q^2,...Q^k]}$ is a System of Automaton Parts for NFA $A = (Q, \Sigma, \delta, q_0, F)$. The System $A/_{[Q^1,Q^2,...Q^k]}$ is called *centralised* if for any set of states $Q^j, j \in \langle 1; k\rangle$ it holds:

1. $\forall i \in \langle 1; k\rangle, i \neq j : (Q^i \cap Q^j) = \emptyset$
2. $\forall i \in \langle 1; k\rangle, i \neq j : (Q^i_{in} \subseteq Q^j_{out})$
3. $\forall i \in \langle 1; k\rangle, i \neq j : (Q^i_{out} \subseteq Q^j_{in})$

Then $A/_{Q^j}$ is called a *central part* or a *central item* of the centralised system $A/_{[Q^1,Q^2,...Q^k]}$.

As the System of Automaton Parts $A/_{[Q^N,Q^{nca}_1,...Q^{nca}_k]}$, which is created from k sets of states without collisions $Q^{nca}_1, ...Q^{nca}_k$ and the set $Q^N = Q \setminus \bigcup_{i=1}^{k} Q^{nca}_i$ is not centralised, we propose a new algorithm to transform System of Automaton Parts $A/_{[Q^N,Q^{nca}_1,...Q^{nca}_k]}$ to the centralised System $A/_{[Q^{c_N},Q^{c_1},...Q^{c_k}]}$ with central part $A/_{Q^{c_N}}$. The algorithm constists of the following four steps:

1. Let $\forall i \in \langle 1; k\rangle, i \neq r : Q^{c_i} = Q^{nca}_i \setminus Q^{nca}_r$
2. Let $Q^{c_N} = Q^N$
3. For all $i \in \langle 1; k\rangle, i \neq r$ do:
   (a) $Q^{c_N} = Q^{c_N} \cup Q^{c_i}_{out}$
   (b) $\forall j \in \langle 1; k\rangle, j \neq r : Q^{c_j} = Q^{c_j} \setminus Q^{c_i}_{out}$
4. The System $A/_{[Q^{c_N},Q^{c_1},Q^{c_2},...Q^{c_k}]}$ is centralised and $A/_{Q^{c_N}}$ is the central part.

The algorithm moves output states of all parts $A/_{Q^{c_i}}$, $i \in \langle 1; k\rangle$, to the central part $A/_{Q^{c_N}}$. Then all outgoing transitions from part $A/_{Q^{c_i}}$, $i \in \langle 1; k\rangle$ can go only to the central part and all conditions for centralised System of Parallel Automaton Parts must hold.

5. NFA Split Architecture

We have introduced the algorithm to find set of states without collisions and the System of Automaton Parts, which corresponds to the division of the automaton according to the multiple subsets of states. The created model of automaton parts can be utilized for mapping of NFA to the FPGA. Using the proposed algorithm, we can find in the automaton $A = (Q, \Sigma, \delta, q_0, F)$ k sets of states without collision $Q^{nca}_1, ...Q^{nca}_k$ and split the automaton A to k deterministic parts $A/_{Q^{nca}_i}$, $i \in \langle 1; k\rangle$ and one nondeterministic part $A/_{Q^N}$, where $Q^N = Q \setminus \bigcup_{i=1}^{k} Q^{nca}_i$. Sets of states Q^{nca}_i and Q^N determine system of parallel automaton parts $A/_{[Q^N,Q^{nca}_1,...Q^{nca}_k]}$, which can be transformed to the centralised system in order to reduce the complexity in communication between parts. For the centralised system, we propose new NFA Split architecture which can be used for mapping of the model to the FPGA technology in order to store part of the transition table in the memory instead of FPGA logic. The architecture is shown in Figure 3.

Nondeterministic part $A/_{Q^N}$ is mapped to FPGA logic as an nondeterministic unit (NU) and deterministic parts $A/_{Q^{nca}_0}, A/_{Q^{nca}_1}, ...A/_{Q^{nca}_k}$ are mapped to deterministic finite automaton units (DU). DU works like a DFA: it preserves the current state and calculates the next state according to the transition table stored in the memory. In addition to DFA, it must *(i)* be able to activate states in NU, *(ii)* support inactive state (no DU state is active) and *(iii)* be able to activate any state from NU.

The NU has to deal with collision states. Therefore the transition table is mapped to FPGA logic using architecture presented by Clark [6], where nondeterministic behavior is solved by fine-grained parallel processing in

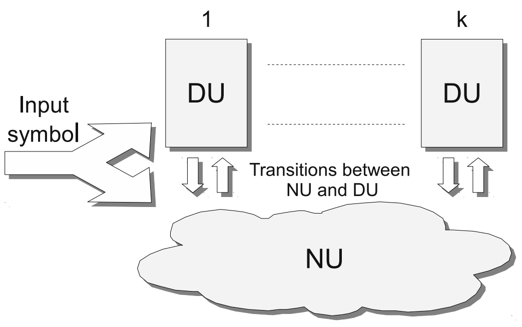

Figure 3: NFA Split architecture consists of processing units which correspond to division of the automaton to deterministic and nondeterministic parallel parts

FPGA logic. States are represented by flip-flops and transitions by combinational logic, which is mapped to look-up tables. A shared decoder is used to convert input symbols to one-hot encoding because it simplifies next-state logic and reduces the amount of consumed resources.

The DU architecture is shown in Figure 4. It consists of a memory (TMEM) to store the transition table, a unit to calculate the next state (NSU), a collision detection unit (CDU) and Input Encoder and Output Decoder for communication with NU.

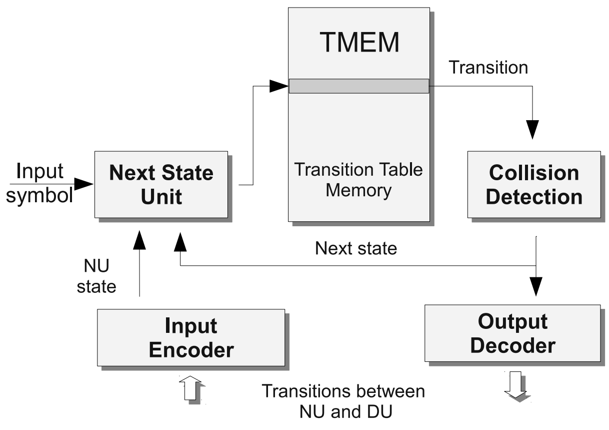

Figure 4: The Architecture of DU consists of a transition table memory, a next state unit, a collision detection unit and two units for communication with NU.

The current state is stored in the NSU and updated to the next state every clock cycle. The next state is calculated from the input symbol, current state and stored transitions in the TMEM. First, the current state and the input symbol are added in order to get an address to TMEM. The address is a pointer to the transition which is read and passed to CDU to check its validity. For a valid transition, the current state is changed to the next state which is stored inside the transition. If the transition is not valid, the DU is switched to the IDLE state.

States without collisions have usually sparse rows in the transition table. As rows contain only a few transitions to next states, we can overlap sparse rows and store all transitions in a smaller memory. If rows are overlapped, we need to recognize which transition belongs to which state. Therefore every transition in the TMEM contains together with the next state value also the value of the input symbol. If the transition contains a different symbol compared to the input, then the transition is for a different state and consequently no transition is defined for the current state.

As every two states s_i a s_j have a different value $s_i \neq s_j$, it is guaranteed that collision between overlapped rows can be reliably detected only by the value of the input symbol. The current state does not have to be stored in every transition to detect the collision. Let us suppose that a collision occurs and is not detected by the stored symbol v_{sym}. Then assume there are two different states s_i and s_j and one input symbol v_{sym}. For the collision, the following equation $s_i + v_{sym} = s_j + v_{sym}$ must hold, but it means that $s_i = s_j$, which contradict the assumption $s_i \neq s_j$.

The overlapping can decrease memory requirements, but it is necessary to find a good placement of rows in the memory. Therefore we designed an algorithm which uses a heuristic to find suboptimal overlapping of transition table rows in the memory. The algorithm consists of the following consecutive steps:

1. Sort all Q^{nca} states according to the number of outgoing transitions.
2. Select state $q \in Q^{nca}$ with the largest number of outgoing transitions, map it to the memory and remove it from Q^{nca}.
3. For the selected state q, find a free place in the memory. All possible locations need to be tested from the starting address. If the row with q cannot fit into the memory, memory size is increased.
4. If $Q^{nca} \neq \emptyset$ then go back to point 2.

As most of the states have only a few outgoing transitions, the proposed algorithm can efficiently overlap all rows and map all states without collisions to memory with overhead less than 10 %.

Both the NU and the DU can activate states in the other unit. Communication between both units is ensured by Input Encoder and Output Decoder. The architecture of both units is shown in Figure 5. Encoder and Decoder are used primarily to convert state values between one-hot and binary encoding. Encoded or decoded values are then issued to DU or NU.

If the DU has to activate one or more states in the NU, a value of state is converted by Output Decoder to signal which represents the state in NU. As the state value has usually less than 16 bits, the decoding is fast and consumes only a few LUTs. An activation of a state in the DU from the NU is more complicated, because transitions to the DU can go to many states which has to be converted to the binary value of the next state.

For many target states, the encoding logic has to convert many inputs to one binary value. It means that many LUTs can be in cascade and maximum frequency of DU can be affected. In order to speed-up the encoding, we use flip-flops to represents also states which are a target

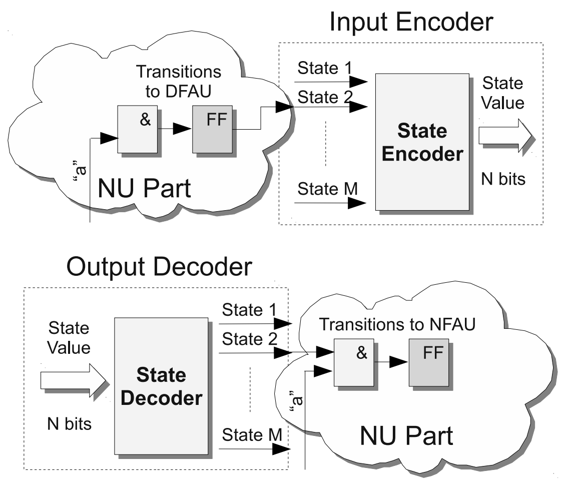

Figure 5: The architecture of Input Encoder and Output Decoder which converts values of states between binary and one-hot encoding.

of transitions to the DU. The encoder is then pipelined in two consecutive stages (state update and encoding). In the first stage, only the next state is calculated, but using retiming a part of the encoder is moved from the second stage to the first pipeline stage and frequency is not decreased.

6. Experimental Results

The proposed architecture was evaluated on various sets of regular expressions. We used regular expressions from L7 decoder [13] and seven different modules of Snort [17]. First, we evaluated the algorithm that computes the set of states without collision and splits the automaton to multiple deterministic and one nondeterministic parts. For every set of reugular expressions, NFA has been created and the proposed algorithm was used to create $k = 8$ non-collision sets of states $Q_1^{nca}, Q_2^{nca}, ..., Q_8^{nca}$. The measured results are shown in Table 3. The table contains the amount of all NFA states Q, the size of non-collision sets $Q_1^{nca}, Q_2^{nca}, ..., Q_8^{nca}$ and the size of Q_N set, which contains remaining states $Q_N = Q \setminus \bigcup_1^8 Q_i^{nca}$

We can see in Table 3 that the proposed algorithm was able to find a non-collision set Q_1^{nca} with 61.4 % of states in average and a non-collision set with 83.6 % of states for the best case. For $A/_{[Q_1^{nca},...Q_8^{nca}]}$, the amount of states represented by non-collision sets $Q_1^{nca}, Q_2^{nca}, ...Q_8^{nca}$ was increased to 84.7 % in average.

We implemented the proposed NFA Split architecture in VHDL together with program for mapping nondeterministic and deterministic parts of the automaton to NU and DU units. Both units were created to process one byte per clock cycle. Then we measured the utilization of Xilinx Virtex-5 LX155T FPGA resources for the proposed architecture and compared the results with mapping of NFA to FPGA logic presented by Clark [6]. Table 4 contains results for NFA Split architecture with one DU, where is mapped deterministic part $A/_{Q_1^{nca}}$ to DU. We can see that the NFA Split architecture with one DU reduces the amount of consumed look-up table to 66.8% and flip-flops to 43.3% in average for all selected sets of regular expressions only at the cost of a few kilobytes of memory, which can be implemented by BlockRAMs. We analysed also

Rule set	Clark et al. LUT [-]	Clark et al. FF [-]	NFA Split LUT [-]	NFA Split FF [-]
L7 decoder	1538	836	1231	237
Snort (1)	4680	4043	2466	821
Snort (2)	2965	876	1883	374
Snort (3)	1637	555	1370	261
Snort (4)	2436	1392	2233	924
Snort (5)	2807	1099	1969	368
Snort (6)	2680	1097	2259	543
Snort (7)	10314	2812	3393	1439

Table 4: The utilization of a Xilinx Virtex-5 LX155T FPGA resources for the proposed NFA Split architecture with one DU.

Rule set	#Tr [-]	Overhead [-]	Overhead [%]	Memory [B]	Memory [BR]
L7 dekodér	2622	65	2.42	9068	6
Snort (1)	18226	109	0.59	61880	27
Snort (2)	15909	90	0.56	53996	24
Snort (3)	7290	133	1.79	25052	12
Snort (4)	4468	457	9.28	16621	9
Snort (5)	6481	12	0.18	21913	12
Snort (6)	8273	174	2.06	28508	15
Snort (7)	6606	26	0.39	22383	12

Table 6: Memory utilization of NFA Split architecture with one DU.

FPGA logic utilization for NFA Split architecture with multiple DUs. The results are shown in Table 5. We can see that multiple DUs can further reduce the amount of consumed FPGA resources, but the reduction is not as significant as for the first DU. The reason is an exponential fall of non-collision sets size with k. While the first non-collision set Q_1^{nca} contains 61.5 % of states in average, it is only 10.5 % of states for the second set Q_2^{nca}.

For NFA Split architecture with one DU, we evaluated the efficiency of mapping NFA transitions to memory using the proposed algorithm. The results are shown in Table 6. The table contains in column #Tr the amount of all transition in the deterministic part $A/_{Q_1^{nca}}$, which corresponds to the minimal representation of $A/_{Q_1^{nca}}$ in memory words. Last two columns contain memory requirements to store $A/_{Q_1^{nca}}$ transition table using the proposed algorithm, which tries to find a good overlapping of transition table rows. The memory requirements are in bytes and FPGA BlockRAMs (BR). The overhead of the algorithm is in the third and fourth column. We can see that the proposed algorithm has an overhead less than 10 % in comparison to minimal $A/_{Q_1^{nca}}$ representation.

7. Conclusion

In the paper we propose new NFA Split architecture which reduces the amount of consumed FPGA resources in order to match a large set of regular expressions at multi-gigabit speed. The proposed reduction uses model of nondeterministic and deterministic automaton for efficient mapping of regular expressions to FPGA. We introduced an algorithm which is able to find non-collision sets of states, split an automaton to one nondeterministic and multiple

Rule set	NFA k Split									
	Q [-]	Q_1^{nca} [%]	Q_2^{nca} [%]	Q_3^{nca} [%]	Q_4^{nca} [%]	Q_5^{nca} [%]	Q_6^{nca} [%]	Q_7^{nca} [%]	Q_8^{nca} [%]	Q^N [%]
L7 dekodér	806	78.4	7.7	4.0	1.7	1.9	0.7	0.5	0.6	4.5
Snort (1)	3888	83.6	6.2	2.6	1.3	0.8	0.7	0.7	0.4	3.7
Snort (2)	819	63.7	8.5	5.0	2.8	2.2	2.0	1.6	1.5	12.7
Snort (3)	527	59.8	7.0	4.9	3.0	2.1	2.1	2.1	2.1	16.9
Snort (4)	1344	35.9	8.8	5.0	2.3	1.6	0.8	0.7	0.5	44.4
Snort (5)	1060	70.7	10.8	5.4	3.5	3.1	1.7	0.9	0.8	3.1
Snort (6)	1038	56.9	18.8	7.5	4.7	3.2	2.4	2.3	0.5	3.7
Snort (7)	2774	53.4	28.4	5.4	4.5	2.3	2.1	1.1	0.9	1.9

Table 3: The size of NFA and non-collision sets found by proposed algorithm.

Rule set	NFA k Split							
	k=2		k=3		k=4		k=8	
	LUT	FF	LUT	FF	LUT	FF	LUT	FF
L7 dekodér	1364	183	1468	159	1495	153	1669	156
Snort(1)	2743	589	2866	497	2990	456	3222	388
Snort(2)	1853	317	1922	289	1938	278	2147	256
Snort(3)	1283	235	1312	220	1330	213	1575	204
Snort(4)	2170	814	2237	755	2294	732	2489	714
Snort(5)	2168	262	2282	215	2412	186	2542	148
Snort(6)	2378	369	2427	299	2442	258	2586	203
Snort(7)	3778	720	4157	578	4370	461	4786	315

Table 5: The utilization of Xilinx Virtex-5 LX155T FPGA resources for the proposed NFA Split architecture for multiple DUs (k=2, k=3, k=4 and k=8.)

deterministic parts and map significant part of the transition table to a memory with a single access port. As it can be seen in Table 3, the proposed algorithm was able to find non-collision sets with 61.4 % of states in average and a non-collision set with 83.6 % of states for the best case. Moreover, the amount of states represented by eight non-collision sets was 84.7 % in average for all sets of regular expressions. We measured the amount of consumed logic resources for the NFA Split architecture with one DU on the Virtex-5 LX155T FPGA and compared the results with Clark mapping of NFA to FPGA. The proposed NFA Split architecture with one DU reduces the amount of consumed look-up tables to 66.8 % and flip-flops to 43.3 % in average for all selected sets of regular expressions only at the cost of a few kilobytes of memory which can be easily implemented by BlockRAMs. NFA Split architecture with multiple DUs can further reduce the amount of consumed FPGA resources, but the reduction is not as significant as for the first DU. The reason is an exponential fall of a non-collision set size with k. As less FPGA resources is utilized, more regular expressions can be supported.

We introduced also an efficient mapping of a transition table to a memory. As deterministic part of the automaton has usually a sparse transition table, we overlapped rows to save space in memory and proved that only input symbols can always detect collisions between two different rows. Therefore transitions can be stored in a small memory with fast access. As we can see in Table 6, the overlapping of rows has low memory requirements for all sets of regular expressions. The proposed algorithm needed only 10 % overhead in comparison to the minimal representation of the transition table.

The presented algorithms and architecture is able to split NFA to multiple deterministic and nondeterministic part. In future work, we want to explore if non-collision sets can be increased by partial determinisation of the automaton. Concurrently, we want to improve mapping of automaton parts to FPGA in order to achieve further reductions of look-up tables and flip-flop registers and support fast change of regular expressions without FPGA reconfiguration.

Acknowledgements.

This research has been partially supported by the Research Plan No. MSM, 6383917201 – Optical National Research Network and its New Applications, Research Plan No. MSM, 0021630528 – Security-Oriented Research in Information Technology and the grant BUT FIT-S-10-1.

References

[1] Z. K. Baker and V. K. Prasanna. A methodology for synthesis of efficient intrusion detection systems on fpgas. In *FCCM '04: Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 135–144, Washington, DC, USA, 2004. IEEE Computer Society.

[2] Z. K. Baker and V. K. Prasanna. Time and Area Efficient Pattern Matching on FPGAs. In *FPGA '04: Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays*, pages 223–232, New York, NY, USA, 2004. ACM Press.

[3] M. Becchi and P. Crowley. A hybrid finite automaton for practical deep packet inspection. In *Proceedings of the International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*, New York, NY, December 2007. ACM.

[4] Y. H. Cho and W. H. Mangione-Smith. Deep Packet Filter with Dedicated Logic and Read Only Memories. In *12th IEEE*

Symposium on Field-Programmable Custom Computing Machines (FCCM 2004), pages 125–134, Napa, CA, 2004.

[5] C. Clark and D. Schimmel. Efficient Reconfigurable Logic Circuits for Matching Complex Network Intrusion Detection Patterns. In *Field Programmable Logic and Application, 13th International Conference*, pages 956–959, Lisbon, Portugal, 2003.

[6] C. R. Clark and D. E. Schimmel. Scalable Pattern Matching for High-Speed Networks. In *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 249–257, Napa, California, 2004.

[7] S. Dharmapurikar, P. Krishnamurthy, T. S. Sproull, and J. W. Lockwood. Deep Packet Inspection using Parallel Bloom Filters. *IEEE Micro*, 24(1):52–61, 2004.

[8] S. Dharmapurikar and J. Lockwood. Fast and scalable pattern matching for content filtering. In *ANCS '05: Proceedings of the 2005 ACM symposium on Architecture for networking and communications systems*, pages 183–192, New York, NY, USA, 2005. ACM.

[9] R. W. Floyd and J. D. Ullman. The compilation of regular expressions into integrated circuits. *J. ACM*, 29(3):603–622, 1982.

[10] J. Koziol. *Intrusion Detection with Snort*. Sams, Indianapolis, IN, USA, 2003.

[11] S. Kumar, B. Chandrasekaran, J. Turner, and G. Varghese. Curing regular expressions matching algorithms from insomnia, amnesia, and acalculia. In *ANCS '07: Proceedings of the 3rd ACM/IEEE Symposium on Architecture for networking and communications systems*, pages 155–164, New York, NY, USA, 2007. ACM.

[12] S. Kumar, S. Dharmapurikar, F. Yu, P. Crowley, and J. Turner. Algorithms to accelerate multiple regular expressions matching for deep packet inspection. In *SIGCOMM '06: Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 339–350, New York, NY, USA, 2006. ACM.

[13] L7 Filtr. Project WWW Page. `http://l7-filter.sourceforge.net/`, 2010.

[14] V. Paxson, K. Asanović, S. Dharmapurikar, J. Lockwood, R. Pang, R. Sommer, and N. Weaver. Rethinking hardware support for network analysis and intrusion prevention. In *HOTSEC'06: Proceedings of the 1st USENIX Workshop on Hot Topics in Security*, pages 11–11, Berkeley, CA, USA, 2006. USENIX Association.

[15] R. Sidhu and V. K. Prasanna. Fast regular expression matching using fpgas. In *FCCM '01: Proceedings of the the 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 227–238, Washington, DC, USA, 2001. IEEE Computer Society.

[16] R. Sidhu and V. K. Prasanna. Fast Regular Expression Matching using FPGAs. In *Proceedings of the 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2001)*, pages 227–238, April 2001.

[17] Snort. Project WWW Page. `http://www.snort.org/`, 2010.

[18] I. Sourdis and D. Pnevmatikatos. Pre-decoded cams for efficient and high-speed nids pattern matching. In *FCCM '04: Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 258–267, Washington, DC, USA, 2004. IEEE Computer Society.

[19] I. Sourdis and D. N. Pnevmatikatos. Fast, Large-Scale String Match for a 10Gbps FPGA-Based Network Intrusion Detection System. In *Field Programmable Logic and Application, 13th International Conference*, pages 880–889, Lisbon, Portugal, 2003.

[20] L. Tan, B. Brotherton, and T. Sherwood. Bit-split string-matching engines for intrusion detection and prevention. *ACM Trans. Archit. Code Optim.*, 3(1):3–34, 2006.

[21] L. Tan and T. Sherwood. Architectures for bit-split string scanning in intrusion detection. *IEEE Micro*, 26(1):110–117, 2006.

[22] F. Yu, Z. Chen, Y. Diao, T. V. Lakshman, and R. H. Katz. Fast and memory-efficient regular expression matching for deep packet inspection. In *ANCS '06: Proceedings of the 2006 ACM/IEEE symposium on Architecture for networking and communications systems*, pages 93–102, New York, NY, USA, 2006. ACM.

[23] Zachary K. Baker and Viktor K. Prasanna. Automatic Synthesis of Efficient Intrusion Detection Systems on FPGAs. In *Proceedings of the 14th Annual International Conference on Field-Programmable Logic and Applications (FPL '04)*, 2004.

Selected Papers by the Author

J. Kořenek and V. Košař. Efficient mapping of nondeterministic automata to fpga for fast regular expression matching. In *Proceedings of the 13th IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems DDECS 2010*, page 6. IEEE Computer Society, 2010.

J. Kořenek and V. Puš. Memory optimization for packet classification algorithms in fpga. In *Proceedings of the 13th IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems*, pages 297–300. IEEE Computer Society, 2010.

J. Kořenek and P. Kobierský. Intrusion detection system intended for multigigabit networks. In *2007 IEEE Design and Diagnostics of Electronic Circuits and Systems*, pages 361–364. IEEE Computer Society, 2007.

J. Kořenek and M. Košek. Flowcontext: Flexible platform for multigigabit stateful packet processing. In *2007 International Conference on Field Programmable Logic and Applications*, pages 804–807. IEEE Computer Society, 2007.

J. Kořenek and V. Puš. Memory optimization for packet classification algorithms. In *Proceedings of the 5th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, Association for Computing Machinery, pages 165–166. Association for Computing Machinery, 2009.

P. Kobierský, J. Kořenek, and L. Polčák. Packet header analysis and field extraction for multigigabit networks. In *Proceedings of the 2009 IEEE Symphosium on Design and Diagnostics of Electronic Circuits and Systems*, pages 96–101. IEEE Computer Society, 2009.

V. Puš and J. Kořenek. Fast and scalable packet classification using perfect hash functions. In *Proceeding of the ACM/SIGDA international symposium on Field programmable gate arrays*, Association for Computing Machinery, pages 229–236. Association for Computing Machinery, 2009.

T. Martínek, T. Málek, and J. Kořenek. Gics: Generic interconnection system. In *2008 International Conference on Field Programmable Logic and Applications*, pages 263–268. IEEE Computer Society, 2008.

T. Martínek, P. Zemčík, and J. Kořenek. FPGA-based platform for network applications. In *Proc. of 8th IEEE Design and Diagnostic of Electronic Circuits and Systems Workshop*, pages 194–197. University of West Hungary, 2005.

T. Martínek, J. Kořenek, and J. Novotný. Network monitoring adaptor for 10gbps technology using FPGA. In *CESNET Conference 2006 Proceedings*, pages 143–151. CESNET National Research and Education Network, 2006.

J. Kaštil, J. Kořenek, and O. Lengál. Methodology for fast pattern matching by deterministic finite automaton with perfect hashing. In *12th EUROMICRO Conference on Digital System Design DSD 2009*, pages 823–289. IEEE Computer Society, 2009.

J. Kaštil and J. Kořenek. Hardware accelerated pattern matching based on deterministic finite automata with perfect hashing. In *Proceedings of the 13th IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems DDECS 2010*, pages 149–152, 2010.

M. Žádník, J. Kořenek, O. Lengál, and P. Kobierský. Network probe for flexible flow monitoring. In *Proc. of 2008 IEEE Design and Diagnostics of Electronic Circuits and Systems Workshop*, pages 213–218. IEEE Computer Society, 2008.

M. Žádník, T. Pečenka, and J. Kořenek. Netflow probe intended for high-speed networks. In *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL05)*, pages 695–698. IEEE Computer Society, 2005.

D. Antoš and J. Kořenek. String matching for IPv6 routers. In *SOFSEM 2004: Theory and Practice of Computer Science*, pages 205–210, 2004.