Irregular Reconfigurable CAM Structures for Firewall Applications

T.K. Lee, S. Yusuf, W. Luk, M. Sloman, E. Lupu, and N. Dulay

Department of Computing,
Imperial College, 180 Queen's Gate, London SW7 2BZ, UK
{tkl97,sy99,w.luk,m.sloman,e.c.lupu,n.dulay}@doc.ic.ac.uk

Abstract. Hardware packet-filters for firewalls, based on content-addressable memory (CAM), allow packet matching processes to keep in pace with network throughputs. However, the size of an FPGA chip may limit the size of a firewall rule set that can be implemented in hardware. We develop two irregular CAM structures for packet-filtering that employ resource sharing methods, with various trade-offs between size and speed. Experiments show that the use of these two structures are capable of reduction, up to 90%, of hardware resources without losing performance.

1 Introduction

FPGA-based firewall processors have been developed for high-throughput networks [3, 6, 7]. Such firewall processors must be able to carry out packet matching effectively based on filter rules. Each filter rule consists of a set of logical operations on different fields of an input packet header. A 'don't care' condition indicates that a field can match any value.

Content-Addressable Memory (CAM) is a searching device that consists of an array of storage locations. A search result can be obtained in constant time through parallel matching of the input with the data in the memory array. CAM based hardware packet-filters [4] are fast and support various data widths [3]. However, the size of an FPGA may limit the number of filter rules that can be implemented in hardware [7]. We describe two hardware structures that employ resource sharing methods to reduce hardware resource usage for packet-filtering firewalls. Resource usage reduces approximately linearly with the degree of grouping of the filter rules in a rule set. These two structures, when applied to CAM based packet-filters, offer various trade-offs between speed and size under different situations involving parallel and pipelined implementations. The contributions described in this paper include:

1. two hardware irregular CAM structures for implementing filter rules;
2. a strategy to generate hardware firewall processors;
3. an evaluation of the effectiveness of the irregular CAM structures, comparing them against regular CAMs.

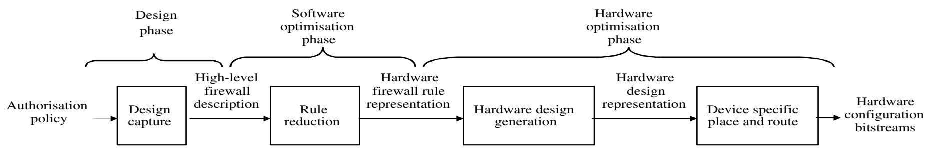

Design phase
Software optimisation phase
Hardware optimisation phase
Authorisation policy
Design capture
High-level firewall description
Rule reduction
Hardware firewall rule representation
Hardware design generation
Hardware design representation
Device specific place and route
Hardware configuration bitstreams

Fig. 1. An overview of our development framework for reconfigurable firewall processors. There are three main phases: the *design phase*, the *software optimisation phase*, and the *hardware optimisation phase.*

The rest of the paper is organised as follows. Section 2 gives an overview of our design framework. Section 3 describes our hardware structures for filter rules. Section 4 outlines the design generation. Section 5 evaluates the performance of our approach in terms of speed and size, and Section 6 provides a summary of current and future work.

2 Framework Overview

As shown in Figure 1, our framework for developing reconfigurable-hardware packet filtering firewalls consists of three main phases: the design phase, the software optimisation phase, and the hardware optimisation phase.

During the *design phase*, the requirements of a firewall are captured as a high-level description. We use a subset of Ponder [2], a policy specification language, to create our firewall description language [5]. This firewall description uses Ponder's parameterised types and the concept of domains. Our high-level firewall description supports abstraction from details of the hardware implementation. It uses constraints to specify low-level hardware requirements such as placement and partitioning, run-time reconfiguration, timing and size requirements, and hardware software co-operation.

During the *software optimisation phase*, high-level firewall rules are reduced and converted to a hardware firewall rule representation, using parameterised library specifications. We have developed a series of rule reduction steps to reduce hardware usage by employing rule elimination and rule sharing methods [5]. A rule set is divided into a number of groups of hardware filter rules. Sequencing is performed to preserve the ordering and the semantics of a rule set. Reordering and partitioning is conducted to facilitate the grouping process. Each group consists of either a list of rules related by common attributes, or a singleton rule if no sharing with other rules can be found within the same partition.

During the *hardware optimisation phase*, hardware firewall rule representations are converted to a hardware design which is then used to produce the hardware configuration bitstreams for downloading onto an FPGA. The next section describes the irregular CAM structures that we develop, and their use in implementing a rule set in hardware and in facilitating resource sharing.

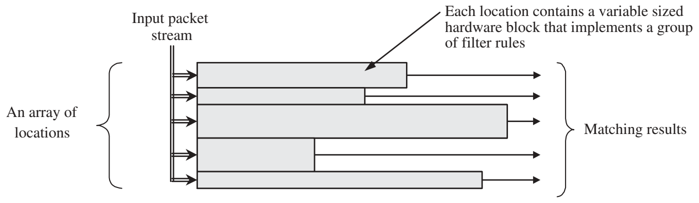

Input packet stream
Each location contains a variable sized hardware block that implements a group of filter rules
An array of locations
Matching results

Fig. 2. A rule set in hardware implemented as an irregular CAM structure. Each location contains a *variable sized* hardware block that implements a *group* of filter rules. An array of hardware blocks together form a CAM structure that supports the whole rule set.

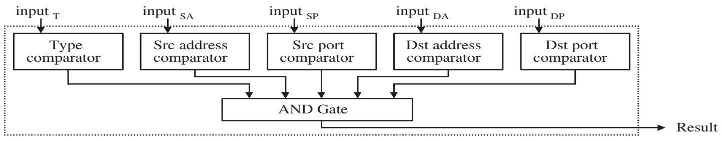


Fig. 3. A hardware block for a group of singleton filter rule. Instead of having a single matching processor as in a regular CAM, our hardware blocks for filter rules contain several field-level hardware comparators that correspond to different fields of an input packet. When a hardware block is instantiated, filter rules that contain 'don't care' fields will have the corresponding hardware comparators eliminated. This example shows the situation when no 'don't care' fields are involved in a filter rule.

3 Architecture of Irregular CAM for Firewall Rules

Our approach for implementing a firewall rule set in hardware is to construct a specialised CAM structure, as shown in Figure 2, to perform packet-filtering. Conventional regular CAM structures store a *single* matching criterion in each memory location. However, instead of having a one-to-one mapping of a filter rule to a CAM location, we construct each CAM location as a hardware block that implements a *group* of filter rules.

A rule set is divided into several groups of filter rules as described in Section 2. Each of these groups is implemented as a hardware block that corresponds to a CAM location. These *variable sized* hardware blocks together then produce an irregular CAM structure that represents the whole filter rule set.

Hardware blocks are instantiated according to the types of grouping and the combinations of field attributes. Figure 3 shows a hardware block for a group of singleton filter rule. It contains several field-level hardware comparators that correspond to different fields of the input. Matching results are obtained as the unified result from all the individual comparators. This design is functionally the same as a regular CAM, except that a regular CAM design will normally use only one comparator for the input data and does not need the AND gate. Separating the matching processor into field-level comparators, however, allows

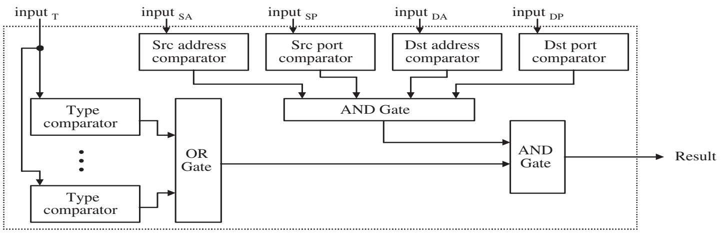


Fig. 4. A hardware block of a group of shared filter rules using the *Siamese Twins* structure. Individual fields of the filter rules having identical data values are shared by using the same hardware comparators. Fields that cannot be shared have their corresponding parts OR-ed together. This example shows that a block is instantiated with all but the *Type*-field comparator being shared.

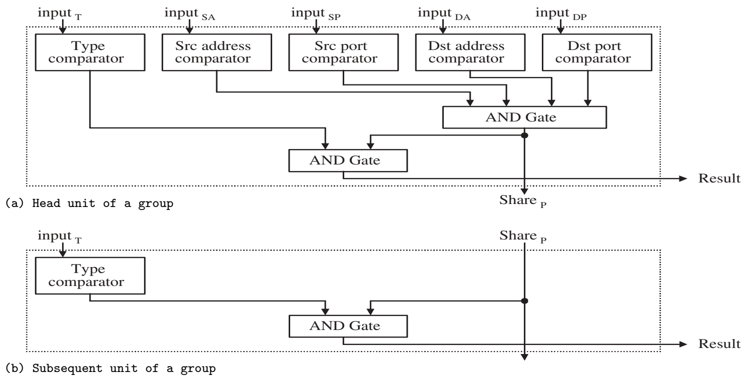


Fig. 5. Hardware blocks of a group of shared filter rules using the *Propaganda* structure. A group of filter rules are sub-divided into a head unit and a number of subsequent units chained to the head. Individual fields of the filter rules having identical data values are shared by using the same hardware comparators in the head unit. The comparison result of the shared fields is propagated from the head unit to each of the subsequent units in the group. Fields that cannot be shared are AND-ed with the propagating result individually. This example shows that the blocks are instantiated with all but the *Type*-field being shared.

us to achieve reduction in resource usage at the expense of introducing a multi-input AND-gate. When a hardware block is instantiated, filter rules that contain 'don't care' fields will have the corresponding hardware comparators eliminated.

Within a group of rules, hardware resources are shared by attributes that are common. There are two levels of sharing: field-level sharing and bit-level

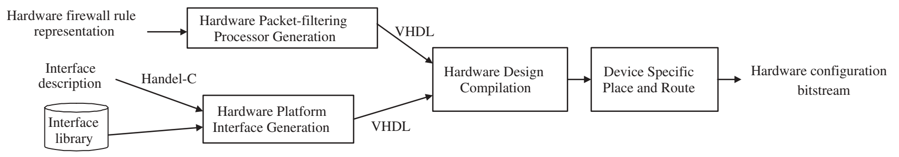

Hardware firewall rule representation
Hardware Packet-filtering Processor Generation
VHDL
Interface description
Handel-C
Interface library
Hardware Platform Interface Generation
VHDL
Hardware Design Compilation
Device Specific Place and Route
Hardware configuration bitstream

Fig. 6. Hardware firewall processor generation. To improve flexibility, the packet-filtering processing unit and the platform-specific interfaces are separately generated.

sharing [5]. We develop two irregular CAM structures: the *Siamese Twins* and the *Propaganda*. They both provide field-level sharing but have different trade-offs between size and speed. A group of shared filter rules are implemented as hardware blocks using either the Siamese Twins structure, which is optimised for area, or the Propaganda structure, which is optimised for speed. Figure 4 and Figure 5 show some examples of hardware blocks for a group of filter rules using the two structures.

In the Siamese Twins structure, the fields of a group of filter rules with identical data values are shared by using the same field-level hardware comparators. Fields that cannot be shared have their corresponding parts OR-ed together. This organisation has the advantage of a simple design, and results in reduction of resource usage by eliminating redundant hardware comparators.

In the Propaganda structure, a group of filter rules are sub-divided into a head unit and a number of subsequent units chained to the head. Individual fields of the filter rules with identical data values are shared by using the same field-level hardware comparators in the head unit. The comparison result of the shared fields is propagated from the head unit to each of the subsequent units in the group. Fields that cannot be shared are AND-ed with the propagating result individually. Filter rules implemented using the Propaganda structure result in a list of hardware blocks joined together. Each filter rule within a group corresponds to a hardware block. The length of the list varies with the number of rules in a group. This is in contrast to the hardware blocks of Siamese Twins or singleton filter rules, where there is only one unit.

4 Hardware Firewall Processor Generation

To generate our hardware firewall processors (Figure 6), we separate the generation of the platform-specific interfaces from the generation of the filtering processor unit. The interfaces are written in the Handel-C language [1], which facilitates porting the design to various hardware platforms.

We design the hardware code generator that takes the hardware firewall rule representation as input, and generate the hardware packet-filtering processor (Figure 6) in VHDL. During the implementation of a CAM location, a hardware block is instantiated according to the attributes of the field in a group of filter rules. These include the combinations of fields that are shared and not shared, and the number of rules in a group. Furthermore, there are structures for replicating and connecting the non-shared fields for a group of rules. All our hardware blocks can be used in both parallel and pipelined mode.

Our implementations target the Xilinx Virtex series FPGAs. We follow the vendor's recommendation [8] of reprogramming lookup tables as Shift Registers (SRL 16). The JBits tool is then used to reconfigure the SRL 16 blocks to desired matching values, for various locations in our irregular CAMs.

5 Performance Evaluation

To analyse the performance of our irregular CAM, we compare implementations that employ the Siamese Twins and the Propaganda structures against those based on regular CAMs. We evaluate the implementations in terms of clock speed and hardware resource consumption.

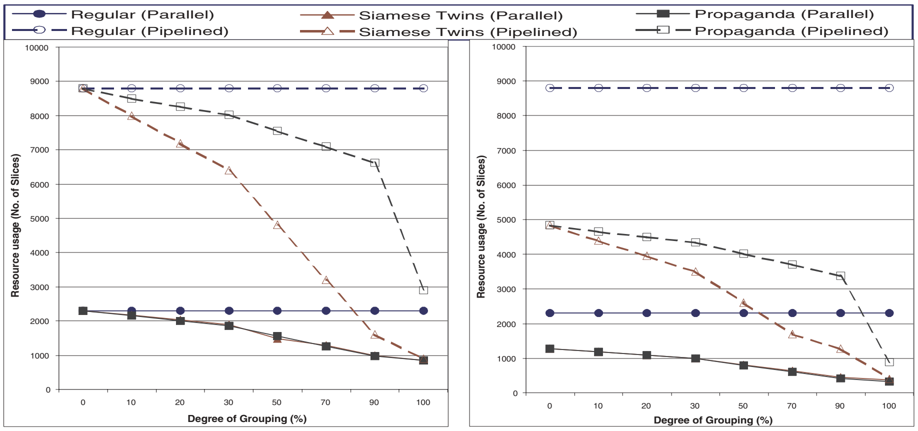
In addition to using rule sets from network sites, we also generate artificial rule sets. Our rule set generator is based on real filter rule sets and covers a wider spectrum of possible real situations as well as some worst-case scenarios. The test data include the effects of 'don't care' fields, the degree to which rules are grouped, and the size of rule sets. For the purpose of the experiments, 'degree of grouping' means the percentage of rules within a rule set that are in a shared group. The resource usage figures include resource to support the I/O to RAM, which is a fixed overhead and is insignificant when compared with the overall resources required by a rule set. All the experiments are performed on a Celoxica RC1000-PP reconfigurable hardware platform that contains a Xilinx Virtex XCV1000 FPGA device.

5.1 Resource Usage

Figure 7 shows the resource usage for rule sets with different degrees of grouping, and the effects of 'don't care' fields. The resource usage of regular CAM remains unchanged as the degree of grouping varies.

When the degree of grouping is at 0% as shown on the left-hand-side of Figure 7 (a) and (b), there is no reduction in resource usage in the case of no 'don't care' fields, but there is around 45% reduction in the case with 'don't care' fields for both the Siamese Twins and the Propaganda structures. A rule set that does not contain any 'don't care' fields in all of its rules is unrealistic. In reality, most rule sets contain a certain amount of 'don't care' fields. This suggests that both Siamese Twins and Propaganda will achieve reduction in resource usage over a regular CAM, whenever a 'don't care' field exists in a rule set.

For the parallel versions, the resource usage of Siamese Twins and Propaganda are about the same as shown in the lower parts of Figure 7 (a) and (b), where their corresponding graphs almost overlap. For the pipelined version, Propaganda uses noticeably more resources than Siamese Twins. This is due to the additional pipeline registers. This suggests that both Siamese Twins and Propaganda are suitable for implementations involving parallel structures. However, if an implementation must involve pipelining and when resource usage is the main concern, Siamese Twins is the preferred choice.



(a) No 'don't care' fields. In this test, none of the filter rules contain 'don't care' fields.

(b) With 'don't care' fields. In this test, each filter rule has one of the address fields as well as one of the port fields assigned as 'don't care'.

Fig. 7. Resource usage versus degree of grouping. Resource usage of both Siamese Twins and Propaganda reduce approximately linearly with the degree of grouping. For the parallel versions, both structures perform almost identically. For the pipelined versions, Siamese Twins is considerably better. Note that for this test, one of the address field is not shared in a group. Since the address field is the largest field in a filter rule, it gives the worst case resource usage for a single non-shared field.

When the degree of grouping is low (less than 10%), the pipelined versions consume 3.8 times more resources then their parallel counterparts. This shows the well-known trade-offs between speed and size. However, when the degree of grouping is high (larger than 70% in the case of no 'don't care' fields, and larger than 50% in the case with 'don't care' fields), the pipelined versions of Siamese Twins consume comparable or fewer resources than the parallel versions of the regular CAM. These two figures correspond to 138% (in the case of no 'don't care' fields) and 188% (in the case with 'don't care' fields) of the speed of the regular CAM. This suggests that, in situations when both size and speed should be optimised, a pipelined version of Siamese Twins can be better than a parallel version of the regular CAM.

5.2 Speed

Figure 8 shows the speed performance for rule sets with different degrees of grouping, and the effects of 'don't care' fields. The maximum operating frequency of regular CAM remains unchanged as the degree of grouping varies.

Results for the parallel versions are shown in the lower parts of Figure 8 (a) and (b). Both Siamese Twins and Propaganda performs approximately the same as the regular CAM. Results for the pipelined versions are shown in the upper parts of Figure 8 (a) and (b). While Propaganda performs comparable

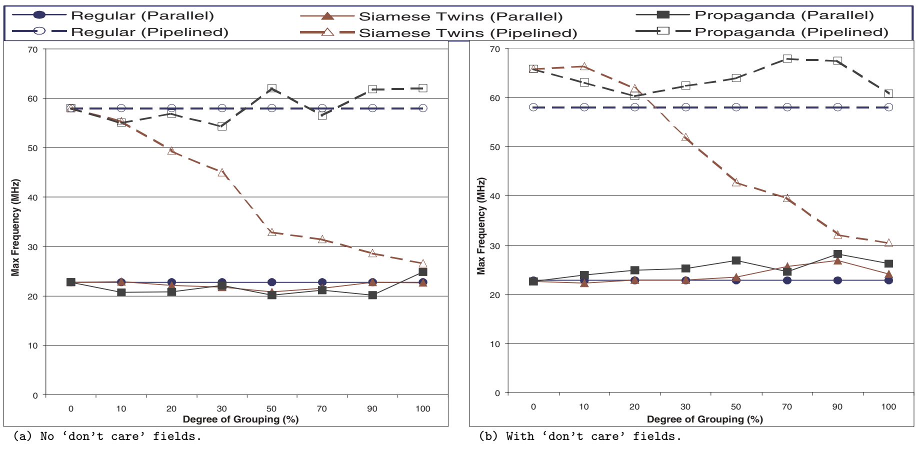


Fig. 8. Speed performance versus degree of grouping. Propaganda consistently achieves comparable performance to the regular CAM. Siamese Twins, while having approximately the same performance as regular CAM in the parallel version, suffers from performance degradation when the degree of grouping increases in the pipelined versions. Note that for this test, all the shared rules using the Siamese Twins structure are grouped into a single CAM location. This produces the highest propagation delay and so the lowest performance.

to or sometimes slightly better than the regular CAM, the Siamese Twins suffers from performance degradation when the degree of grouping increases. This reduction in performance is due to the increased routing and propagation delay of the enlarged OR-structure inside the Siamese Twins. When the degree of grouping is low (less than 10%), both structures are 2.5 times faster than their parallel counterparts. When the degree of grouping is at 100%, the performance of Siamese Twins is reduced by nearly 50% to have similar performance to its parallel counterpart. This suggests that both Siamese Twins and Propaganda are suitable for implementations involving parallel structures. However, if implementations involve pipelining and when speed is also a major concern, Propaganda can be a better choice.

Figure 9 shows that maximum operating frequency is determined not only by the degree of grouping, but also by the maximum group size. For the parallel versions, both Siamese Twins and Propaganda do not vary much with the degree of grouping. For the pipelined versions as shown in the top-left parts of Figure 9 (a) and (b), when the group size is small, the performance of Siamese Twins is comparable to the regular CAM even at 100% degree of grouping. When the group size is large (100 rules/location), its performance decreases by nearly 50%.

The effects of maximum group size suggest that there can be a trade-off between resource utilisation and the maximum operation frequency. In order to avoid performance degradation at a high degree of grouping, one can choose

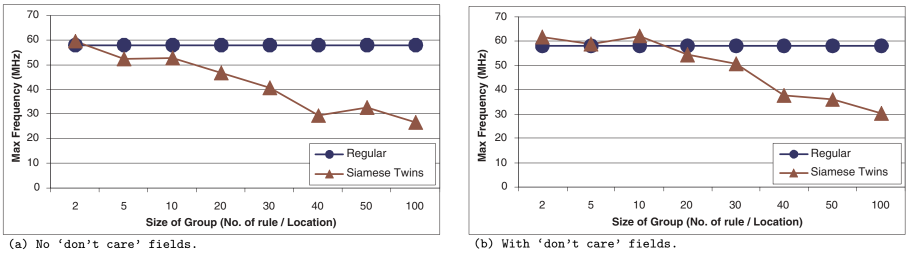


Fig. 9. Speed performance versus maximum group size for pipelined implementations. When the maximum group size is small (less than 20 rules/location in the case of no 'don't care' fields, and less than 30 rules/location in the case with 'don't care' fields) the performance of Siamese Twins and the regular CAM are comparable. In this test, the degree of grouping is always 100%, but the group of shared filter rules are broken down into a number of smaller groups.

to impose either a ceiling group size or a maximum degree of grouping for the pipelined versions of Siamese Twins. For example, groups with number of rules exceeding the ceiling group size can be broken down into several smaller groups. This method can maintain performance, but at the expense of using more hardware resources to implement the additional groups.

5.3 Results Summary

The analysis results are discussed in Section 5.1 and Section 5.2. The maximum reduction in hardware usage and maximum group size before performance degradation are shown respectively in Table 1 and Table 2. For the purpose of the experiments, performance degradation is defined as no more than 10% reduction in speed, when compared to corresponding regular CAMs.

Table 1. Maximum reduction in hardware usage before performance degradation.

		Reduction in hardware usage	Degree of grouping
Siamese Twins	Parallel	84%	100%
	Pipelined	60% with 'don't care'	30%
		18% without 'don't care'	20%
Propaganda	Parallel	84%	100%
	Pipelined	90%	100%

Table 2. Maximum group size before performance degradation (Siamese Twins pipelined).

	Rule / Location	Reduction in hardware usage
With 'don't care'	30	90%
Without 'don't care'	10	85%

6 Conclusion

We have presented the Siamese Twins and the Propaganda irregular CAM structures. These two structures employ resource sharing to reduce hardware usage for packet-filtering firewalls. Experiments show that resource usage reduces approximately linearly to the degree of grouping of the filter rules in a rule set. These two irregular CAM structures offer various trade-offs between speed and size, under different situations involving parallel and pipelined implementations. Both structures are capable of reduction, up to 90%, of hardware resources of regular CAMs without losing performance.

Current and future work includes the use of bit-level sharing to achieve further reduction in hardware usage, and global and local optimisations of irregular CAM using the Siamese Twins and the Propaganda structures.

Acknowledgements

The support of UK Engineering and Physical Sciences Research Council (Grant number GR/R 31409, GR/R 55931 and GR/N 66599), Celoxica Limited and Xilinx, Inc. is gratefully acknowledged.

References

1. Celoxica Limited, *Handel-C v3.1 Language Reference Manual*, http://www.celoxica.com/.
2. N. Damianou, N. Dulay, E. Lupu and M Sloman, “The Ponder Policy Specification Language”, in *Proc. Workshop on Policies for Distributed Systems and Networks*, LNCS 1995, Springer, 2001, pp. 18-39.
3. J. Ditmar, K. Torkelsson and A. Jantsch, “A Dynamically Reconfigurable FPGA-based Content Addressable Memory for Internet Protocol Characterization”, *Field Programmable Logic and Applications*, LNCS 1896, Springer, 2000.
4. P.B. James-Roxby and D.J. Downs, “An Efficient Content-addressable Memory Implementation Using Dynamic Routing”, in *Proc. IEEE Symp. on Field-Programmable Custom Computing Machines*, IEEE Computer Society Press, 2001.
5. T.K. Lee, S. Yusuf, W. Luk, M. Sloman, E. Lupu and N. Dulay, “Compiling Policy Descriptions into Reconfigurable Firewall Processors”, in *Proc. IEEE Symp. on Field-Programmable Custom Computing Machines*, IEEE Computer Society Press, 2003.
6. J.T. McHenry and P.W. Dowd, “An FPGA-Based Coprocessor for ATM Firewalls” in *Proc. IEEE Symp. on Field-Programmable Custom Computing Machines*, IEEE Computer Society Press, 1997.
7. R. Sinnappan and S. Hazelhurst, “A Reconfigurable Approach to Packet Filtering”, *Field Programmable Logic and Applications*, LNCS 2147, Springer, 2001.
8. Xilinx Inc., *Designing Flexible, Fast CAMs with Virtex Family FPGAs*, 1999, http://www.xilinx.com/.