Design and Implementation of Efficient Packet Classification on FPGA for Noc based Designs

Rashmi D
M.Tech, Dept. of Electronics and Communication
Sir.M Visvesvaraya Institute of Technology ,
INDIA

Phanindar Ravi P
Asst.Professor, Dept. of ECE
Sir.M Visvesvaraya Institute of Technology,
INDIA

***Abstract*— Packet classification is a complicated and vital task as the processing of packets should be done at a specified line speed. The packet classification is mainly used by networking equipments to sort incoming packets into flows by comparing their headers values to a list of rules. The packets are placed in the flow determined by the matched rule. In order to decide a packet's priority and the manner in which packet is processed, a flow is used. Packet classification is not a simple task because packets must be processed at a wire speed and tens of thousands of rules is present in the rule sets.**

The hardware accelerator or packet classifier has been processed here uses a modified version of Hypercuts packet classification algorithm and it also uses a new pre-cutting process which reduces the amount of memory needed to save the search structure for the large rule sets so that it is small to fit in the on-chip memory of an Field programmable Gate Array. The contribution of this project is a hardware accelerator or classifier that can classify up to 433 million packets per second at the speed of 138.56 Gb/s when using rule sets containing tens of thousands of rules with a power consumption of only 9.03 W which is low compared to other FPGA based classifiers. The modified Hypercuts algorithm allows higher clock speeds and thus obtaining higher throughputs by removing the need for floating point division to be performed when classifying a packet.

Keywords*— Hardware accelerator, Classifier, high throughput, low power, packet classification, parallel processing, pre-cutting.

I. INTRODUCTION

The necessity of packet classification is considered to be important as the burden of a router is reduced. Initially the task of putting a real strain on the networking equipment has to be inspected and processed to resist the resultant traffic. Existing algorithms still have very low performance, and ternary content addressable memories still have issues in terms of power consumption and chip density. In spite of the large number of techniques explored, there are still new techniques in packet classification that can provide major benefits. Network processors are used to process packets when they pass through a network performs various tasks such as packet classification, packet fragmentation and reassembly, forwarding and encryption. The network processors have placed under increased pressure due to the increased number of tasks that need to be carried out, along with the increase in line rates. This pressure is reduced by the adding extra processing capacity is difficult due to tight power budgets and silicon limitations. Increasing the clock speeds to obtain extra performance is difficult due to physical limitations in the silicon, while writing the software used to control the operation of the network processors becomes difficult due to the increased number of processing cores. Above approaches leads to large increases in power consumption due to the additional transistors needed to increase the number of processing cores and the extra heat generated by raising the clock speed.

Hardware accelerators can also process more data than a general-purpose processor while running at slower clock speeds as they are optimized to carry out particular tasks. A reduction in number of transistors and clock speed leads to large savings in power consumption and area.

The contribution of this project is the design and implementation of an efficient packet classification hardware accelerator on Field programmable gate array for Network on Chip based designs. Packet classification is difficult task because all packets entering a network must be processed at wire speed. This problem becomes very difficult due to rule sets containing many rules are needed, while the large number of services is being provided by network providers. To improve the security, the hardware accelerator used here allows packet classification to be done at the core of a network. It uses several packet classification engines operating in parallel with a shared memory.

The classifier proposed in this project uses a multiple packet classification working in parallel with the shared memory, allowing it to classify packets at the speeds of up to 138.56 Gb/s. This Classifier classifies 433 million packets per second, while using rule sets containing tens of thousands of rules. It implements a modified version of the HyperCuts packet classification algorithm, which breaks a rule set into groups, with each group containing a small number of rules that can be searched linearly. A decision tree is used to guide a packet based on its header values to the correct group to be searched.

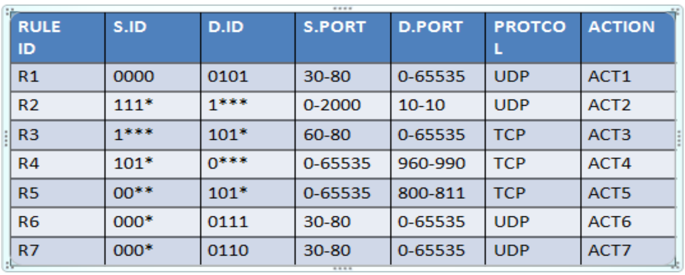
The rest of the paper is organized as follows. Section II describes the Hypercut packet classification. Section III describes the modifications in the Hypercut algorithm. Section IV explains the architecture of the classification engine and the classifier.

II. HYPERCUT PACKET CLASSIFICATION

The fields of a packet's header are the 32 b source and destination IP addresses the 16 b source and destination port numbers, and the 8 b protocol number which are most commonly used to perform packet classification. The easiest way to match these five fields of the header to a rule is to linearly search through the rules one at a time, starting with the highest priority rule and ending with the lowest priority rule, until a match is found. This will result in an unacceptably large worst case processing time, making it difficult to classify packets at the speeds required for the core or even edge of a network. This worst case amount of processing time can be reduced by using the HyperCuts packet classification algorithm. It is a decision tree-based algorithm that builds a search structure that allows incremental updates to a rule set. Search structures that allow incremental updates do not have to be rebuilt each time a rule set has a rule added or deleted. HyperCuts works by breaking a rule set into smaller number of groups, with each group containing a small number of rules suitable for a linear search.

HyperCuts creates this decision tree by taking a geometric view of a rule set, with each rule considered to be a hypercube in hyperspace. The boundaries of each hypercube are defined by the ranges of the rule it represents. The algorithm cuts into this hyperspace by performing cuts to the fields used to define it. Each cut will create sub regions, with each sub region containing the rules whose hypercube overlap. The information regarding the first set of cuts used to divide the hyperspace is stored in the root node of a decision tree. This information includes the number of cuts that are to be performed to each field and the memory location of each of the resulting sub regions. These sub regions are known as the root’s child nodes, with sub regions that contain no rules known as empty nodes. Sub regions whose number of rules does not exceed a user-defined limit are known as leaf nodes. This user-defined limit is known as the binth value. Each leaf node stores one rule group that can be searched linearly. A sub region that contains more rules than is allowed by the binth value is known as an internal node and the space it occupies must be further cut up into smaller sub regions. It also stores the memory locations of the resulting sub regions that is the internal node’s child nodes. An Internal nodes can also have empty, leaf, and internal nodes. The division of the hyperspace into ever-smaller sub regions ends when the number of rules in all sub regions does not exceed the binth value.

The decision tree can be built from the rule set shown in Table 1. The source and destination IP addresses have been reduced from 32 to 4 bits to aid the explanation. The first step in building the decision tree is to decide a value of binth. In this example, binth will be two. The next step involves deciding which dimensions should be used by the root node to cut the hyperspace. This is done by first calculating the number of distinct range specifications for each field.



RULE ID	S.ID	D.ID	S.PORT	D.PORT	PROTOCOL	ACTION
R1	0000	0101	30-80	0-65535	UDP	ACT1
R2	111*	1***	0-2000	10-10	UDP	ACT2
R3	1***	101*	60-80	0-65535	TCP	ACT3
R4	101*	0***	0-65535	960-990	TCP	ACT4
R5	00**	101*	0-65535	800-811	TCP	ACT5
R6	000*	0111	30-80	0-65535	UDP	ACT6
R7	000*	0110	30-80	0-65535	UDP	ACT7

TABLE 1: EXAMPLE RULESET CONTAINING SEVEN RULES

The next step involves trying all combinations of cuts between the chosen dimensions that are less than or equal to 4, with the maximum number of rules stored in a child node for each combination of cuts recorded. The combinations of cuts that can be made to the source and destination IP addresses are [0, 2], [0, 4], [2, 0], [2, 2], and [4, 0]. The combination that results in the smallest maximum number of rules stored in a child node is to cut both the source and destination IP addresses in two.

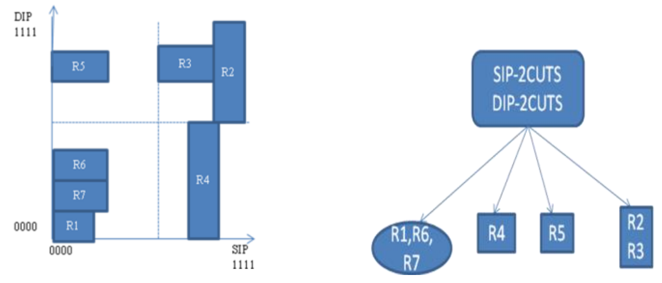


Fig.1. Cuts made to a root node.

Fig.1 shows the decision tree after performing these cuts. It also shows a geometric representation of the source and destination IP addresses, showing the cuts made to the root node (represented by an octagon in the decision tree). It can be seen that these cuts create four sub regions. Three of these sub regions conform to the binth value as they contain two or less rules. This means that they are leaf nodes (represented by rectangles in the decision tree). The fourth sub region contains more rules than the binth value allows. This means that it is an internal node (represented by an oval in the decision tree) that must be cut further.

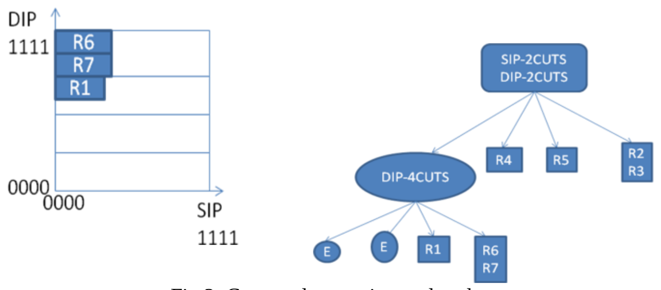


Fig 2. Cuts made to an internal node

Fig. 2 shows the finished decision tree and the cuts performed to the destination IP address when cutting the internal node. It can be seen that two of the sub regions contain no rules which means that they are empty nodes (represented by circles in the decision tree). The remaining two sub regions are stored as leaf nodes.

A. Methods Used to Reduce Memory Usage

The HyperCuts packet classification algorithm uses different heuristics to reduce the amount of memory needed to save a decision tree and the number of memory accesses required to match a rule.

First method is called node merging and it is used to avoid the duplicated storage of identical nodes. Node merging is carried out by first searching the decision tree for leaf nodes that contain the same list of rules. The pointers to these nodes (stored in root and internal nodes) are then modified so that they point to just one of these leaf nodes, meaning that multiple copies do not need to be stored.

A second method is called rule overlap is used to avoid the storage of rules in leaf nodes that can never be matched. A rule can never be matched and is, therefore, removed from a leaf node if the hypercube of a rule with a higher priority completely covers the space it occupies within the leaf node's sub region.

A third method is used to avoid the duplicated storage of rules is called pushing common rule subset upward. These methods stores rules at an internal or root node that would otherwise need to be stored in the internal or root node's entire sub regions.

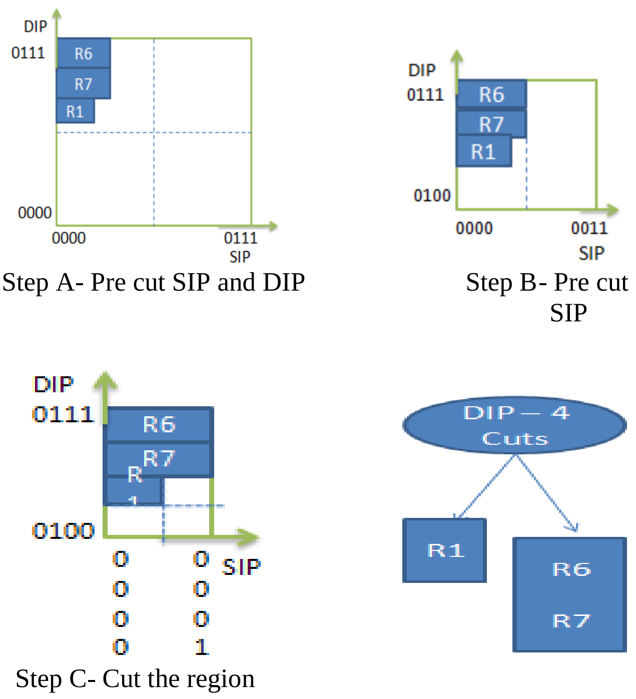
The final method is called region compaction and it is used to aid in the more efficient cutting of the hyperspace. Each node in a decision tree will cover a specific region of the hyperspace. The rules associated with a node may, however, cover a smaller region. Region compaction shrinks the area covered by a node so that it only covers the minimum amount of hyperspace that will cover all rules linked with the node. This means that a smaller region will need to be cut when dividing the hyperspace occupied by a node into sub regions. This could result in lesser cuts, hence memory consumption is reduced.

III. MODIFICATIONS IN THE HYPERCUT ALGORITHM

The HyperCuts algorithm works well when implemented in software. It is not, however, optimized for implementation with dedicated hardware. This section explains the modifications made to the pre-cutting scheme. The pushing common rule subset upward method is not used as it was found during testing of rule sets to make only a fractional reduction in memory usage. It also results in a more complicated search structure that would slow down the classifier as it would have to be able to search root, internal and leaf nodes for matching rules. Pushing common rules upwards can also add extra memory accesses when classifying a packet. This is because a leaf node might still need to be searched even if a matching rule is found at an internal or root node. This is because a leaf node might contain another matching rule with a higher priority. Such a case would mean that the search of the rules at internal or root nodes was unnecessary.

A. PRE-CUTTING SCHEME

A new method for compacting the region to be cut at each internal or root node called pre-cutting is presented here. It uses the same methods employed by the scheme that uses no region compaction when calculating the sub region a packet should traverse to. This scheme only requires an internal or root node to store the number of cuts that must be performed to each field of a packet header and the bits in these fields where the cuts are to be performed. The simplicity of this scheme helps to improve throughput and decrease power consumption. The region that needs to be divided is compacted by recursively cutting all fields in two. This cutting of a specific field in two stops and will not be carried out if it results in rules being contained in more than one sub region. Each precut to a field used to divide the region will halve the number of sub regions that need to be stored and the number of cuts that need to be performed to a packet header when selecting the sub region to traverse to. Each precut to a field also means that the bits which need to be inspected in that field of a packet's header are shifted to the right by one place.



Step A- Pre cut SIP and DIP

Step B- Pre cut SIP

Step C- Cut the region

Fig.3. Compacting of a region using pre-cutting scheme.

Fig. 3 shows an example where pre-cutting is used to compact the area covered by the internal node from the decision tree shown in Fig 2 so that it can be cut more efficiently. The process begins by performing precuts to the source and destination IP addresses as shown in step A, reducing the area that needs to be considered for cutting by 75%. Precuts can be performed to both fields as it results in only one sub region that contains rules. In step B, only the source IP is precut as pre-cutting the destination IP addresses would result in more than one sub region that contains rules. Pre-cutting the source IP address in step B reduces the area that needs to be considered for cutting by another 50%. Finally, in step C no more precuts can be performed so the compacted region is cut in two, with none of the resulting sub regions containing more than two rules. Pre-cutting gives the same effect as the region compaction method used by

HyperCuts in this example, with the number of sub regions that need to be stored reduced from four to two when compared to the method where no region compaction is used.

IV. ARCHITECTURE OF THE CLASSIFICATION ENGINE

The architecture of packet classification engine is shown in Fig 4 which consists of two blocks. The first block is a tree traverser that is used to traverse a decision tree using header information from the packet being classified. The decision tree is traversed until an empty node is reached, meaning that there is no matching rule, or a leaf node is reached. A leaf node being reached will result in the tree traverser passing the packet header and information about the Leaf node reached to the second block known as the leaf node searcher. The leaf node searcher compares the packet header to the rules contained in the leaf node until either a matching rule is found or the end of the leaf node is reached. The leaf node searcher consists of two comparator blocks that work in parallel. This allows two rules to be searched on each memory access, reducing lookup times. Information on the decision tree's root node is stored in registers in the tree traverser, making it possible for the tree traverser to begin classifying a new packet while the previous packet is being compared with rules in a leaf node. This use of pipelining allows for a maximum throughput of one packet every two clock cycles if the decision tree is made up of only a root node and leaf nodes containing no more than two rules.

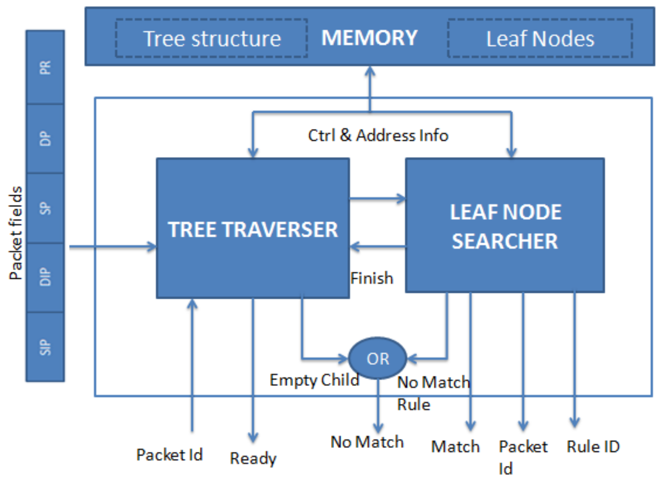
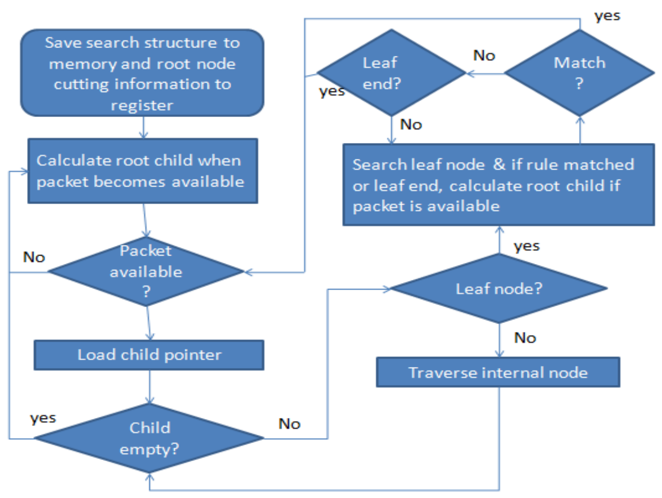


Fig.4. Architecture of the packet classification engine.

The operation of the packet classification engine is explained by the Flowchart shown in Fig 5. The engine has been designed in such a way that it has to traverse a root or internal node in one memory access. It can also search leaf nodes at a rate of two rules per memory access.


Save search structure to memory and root node cutting information to register
Calculate root child when packet becomes available
Packet available ?
No
Load child pointer
Child empty?
yes
No
Leaf end?
yes
No
Match ?
yes
No
Search leaf node & if rule matched or leaf end, calculate root child if packet is available
yes
Leaf node?
No
Traverse internal node
Fig 5. Operation of a packet classification engine.

A. Architecture of the Hardware Accelerator or Classifier

The architecture of classifier or hardware accelerator which is implemented with four classification engines working in parallel is shown in Fig 6.

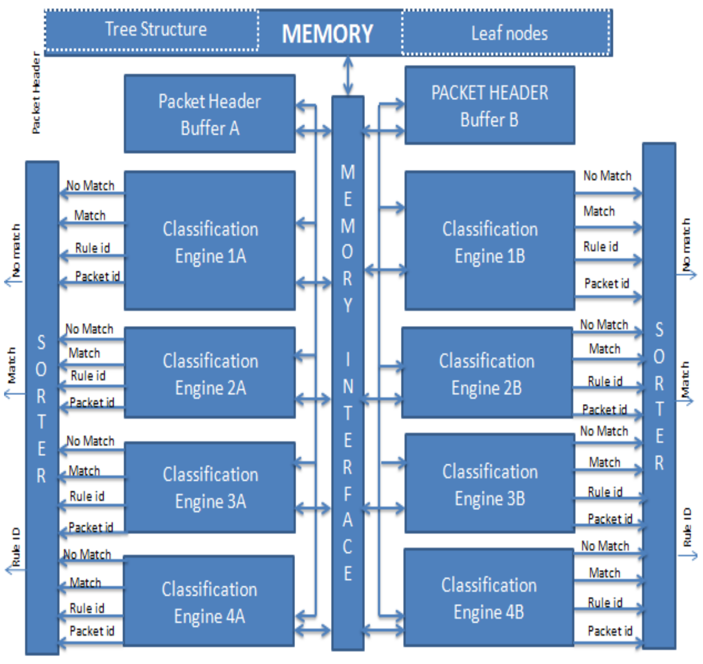


Fig 6. Architecture of Hardware accelerator

The use of multiple engines will help to ensure that the bandwidth of a FPGAs internal memory is better utilized. The use of multiple engines will help to ensure that the bandwidth of a FPGAs internal memory is better utilized. The packet buffer stores the five header fields of the incoming packets. It works on a first come, first served basis, with packets being outputted from the buffer to the packet classification engines in the same order that they were inputted. The buffer also creates a packet ID for each header that is passed to the packet classification engine along with the packet header. The packet ID is used to make sure that the matching rule IDs are outputted by the classifier in the same order that the packet headers were inputted to the system.

The four engines belonging to a classifier run at the same clock speed, with the clock used by each engine 90° out of phase with the clock used by the previous engine. Memory runs at a speed equal to four times that of an engine, ensuring a simple memory interface, with each engine guaranteed access to memory on each of its clock cycles. The memory used is made up of a series of small memory blocks which are connected up so that they act as a continuous memory space. The memory ports of each memory block have their own enable signals. These enable signals are used to reduce power consumption by only activating the memory blocks that are being read from on a given clock cycle. This architecture also allows the splitting of a rule set used to classify packets into groups of four or two in order to reduce the memory consumption and the worst case number of memory accesses needed to classify a packet for rule sets containing a large number of wildcard rules.

The sorter logic block is used to make sure that the matching IDs are outputted in the correct order and that the rule with the highest priority is selected when there are multiple rule matches in the case where rule sets are broken up into groups. The sorter logic block accepts the Match, No Match, Rule ID, and Packet I D signals from each of the packet classification engines. It knows that an engine has finished classifying a particular packet when either the Match or No Match signals have been asserted. The first job the sorter logic block does is to make sure that the rule with the highest priority is selected between engines working in parallel to classify the same packet. This is done by picking the lowest rule ID between packets with the same packet ID. The sorter logic block registers the Match, No Match, and Rule I D signals for a classified packet to a chain of multiplexers and registers in series. The selected register will depend on the packet ID number. The Match, No Match, and Rule ID signals will be registered to the output register if they are next in the sequence of results to be outputted, and stored if not. All stored results are shifted toward the output register each time a result appears that is due to be outputted. This means that the classification results are outputted from the classifier in the same order that the packets were inputted.

V. SIMULATION RESULTS AND ANALYSIS

The classifier can be tested by measuring its logic and memory usage, throughput in terms of Mpps (millions of packets per second), amount of memory it requires when storing the search structures needed to classify packets. The classifier can also be tested by writing verilog programs for the entire classifier design using Xilinx ISE 12.2/13.4. Simulation results are obtained from Modelsim 6.3f which is a very famous commercial simulation tool in electronic industry and is synthesized for Spartan 3(Device XC3S400) FPGA.

The function of the classifier is to classify packets based on the header field values of the incoming packet. In accordance to the above condition, the waveform results are shown above in figure 7. Simulation result in Fig 7 explains input data packet comes through the particular output destination port only when five fields of input data packet matches with the fields of the rule in the leaf node corresponding to that particular output destination port.

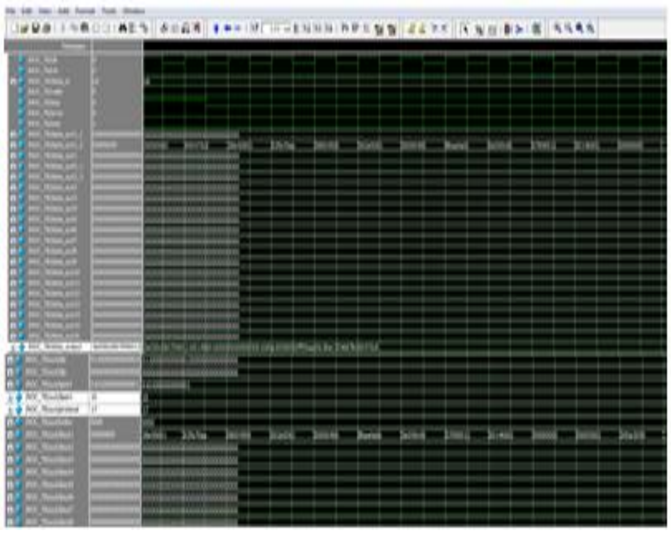
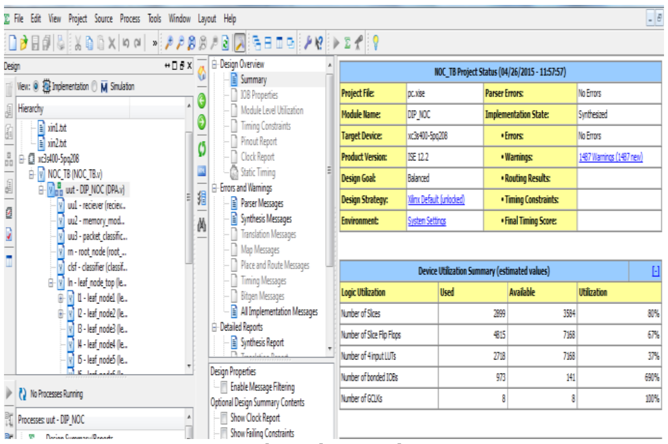


Fig 7. Simulation result



NOC_TB Project Status (04/26/2015 - 11:57:57)			
Project File:	pc.xise	Parser Errors:	No Errors
Module Name:	DIP_NOC	Implementation State:	Synthesized
Target Device:	xc3s400-5pq208	• Errors:	No Errors
Product Version:	ISE 12.2	• Warnings:	1487 Warnings (1487 new)
Design Goal:	Balanced	• Routing Results:	
Design Strategy:	Xilinx Default (unlocked)	• Timing Constraints:	
Environment:	System Settings	• Final Timing Score:	

Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slices	2899	3584	80%
Number of Slice Flip Flops	4815	7168	67%
Number of 4 input LUTs	2718	7168	37%
Number of bonded IOBs	973	141	690%
Number of GCLKs	8	8	100%

Fig 8. HDL synthesis device utilization summary

Fig 8 shows HDL synthesis device utilization summary. From the device utilization summary, our proposed classifier architecture utilizes 80% slices, 67% slice flip flops, 37% four input LUTs, 690% bonded IOBs and it utilizes eight GCLKs.

VI. CONCLUSION

This paper presents a new algorithm and packet classifier or hardware accelerator with enough processing power to allow packet classification to be implemented at the core of the network to improve security. The classifier classifies 433 Mpps (million packets per second) at the speed of up to 138.56 Gb/s by consuming power of 9.03 W which is low compared to other FPGA based classifiers. It worked with rule sets containing tens of thousands of rules at the same 138.56 Gb/s speed. The classifier uses a Hypercut algorithm that has been modified so that it is better suited for hardware implementation.

REFERENCES

[1] *Usage and Population Statistics*. (2012, Jun.) [Online]. Available: http://www.internetworldstats.com/stats.htm

[2] S. Singh, F. Baboescu, G. Varghese, and J. Wang, “Packet Classification using multidimensional cutting,” in *Proc. ACM Special Interest Group Data Commun. Conf.*, Aug. 2003, pp. 213–224.

[3] M. Gupta and S. Singh, “Greening of the internet,” in *Proc. ACM Special Interest Group Data Commun. Conf.*, Aug. 2003, pp. 19–26.

[4] P. Gupta and N. McKeown, “Packet classification using Hierarchical intelligent cuttings,” *IEEE Micro*, vol. 20, no. 1, pp. 34–41, Feb. 2000.

[5] P. Gupta and N. McKeown, “Packet classification on multiple fields,” in *Proc. ACM Special Interest Group Data Commun. Conf.*, Sep. 1999, pp. 147–160.

[6] F. Baboescu and G. Varghese, “Scalable packet classification,” *IEEE/ACM Trans. Netw.*, vol. 13, no. 1, pp. 2–14, Feb. 2005.

[7] T. V. Lakshman and D. Stiliadis, “High-speed policy based Packet forwarding using efficient multi-dimensional range matching,” in *Proc. ACM Special Interest Group Data Commun. Conf.*, Sep. 1998, pp. 203-214

[8] F. Baboescu, S. Singh, and G. Varghese, “Packet classification for core routers: Is there an alternative to CAMs?” in *Proc. IEEE Int. Conf. Comput. Commun.*, Apr. 2003, pp. 53–63.

[9] V. Srinivasan, S. Suri, and G. Varghese, “Packet classification for core routers: Is there an alternative to CAMs?” in *Proc. IEEE Int. Conf. Comput. Commun.*, Apr. 2003, pp. 53–63.

[10] P. Gupta and N. McKeown, “Algorithms for packet Classification using tuple space search,” in *Proc. ACM Special Interest Group Data Commun. Conf.*, Sep. 1999, pp. 135–146.

[11] T. Woo, “A modular approach to packet classification: Algorithms and results,” in *Proc. IEEE Int. Conf. Comput. Commun.*, Mar. 2000, pp. 1213–1222.

[12] A. Kennedy, D. Bermingham, X. Wang, and B. Liu, “Power analysis of packet classification on programmable network processors,” in *Proc. IEEE Int. Conf. Signal Process. Commun.*, Nov. 2007, pp. 1231–1234.

[13] D. E. Taylor and J. S. Turner, “Classbench: A packet Classification bench-mark,” *IEEE/ACM Trans. Netw.*, vol. 15, no. 3, pp. 499–511, Jun. 2007.

[15] B.Vamanan, G. Voskuilen, and T. Vijaykumar, “Efficuts: Optimizing packet classification for memory and throughput.” in *Proc. ACM Special Interest Group Data Commun. Conf.*, Aug. 2010, pp. 207–218.

[16] E. Spitznagel, D.Taylor, and J.Turner, “Packet Classification using extended TCAMs,” in Proc. 11th IEEE Int. Conf. Netw. Protocols,Nov. 2003, pp. 120–131.