# Multi-Engine Packet Classification Hardware Accelerator

Alan Kennedy, Zhen Liu, Xiaojun Wang
School of Electronic Engineering
Dublin City University
Dublin 9, Ireland
{alan.kennedy, liuzhen, wangx} @eeng.dcu.ie

Bin Liu
Department of Computer Science and Technology
Tsinghua University
Beijing, P.R.China
liub@tsinghua.edu.cn

*Abstract*— **As line rates increase, the task of designing high performance architectures with reduced power consumption for the processing of router traffic remains important. In this paper, we present a multi-engine packet classification hardware accelerator, which gives increased performance and reduced power consumption. It follows the basic idea of decision-tree based packet classification algorithms, such as HiCuts and HyperCuts, in which the hyperspace represented by the ruleset is recursively divided into smaller subspaces according to some heuristics. Each classification engine consists of a Trie Traverser which is responsible for finding the leaf node corresponding to the incoming packet, and a Leaf Node Searcher that reports the matching rule in the leaf node. The packet classification engine utilizes the possibility of ultra-wide memory word provided by FPGA block RAM to store the decision tree data structure, in an attempt to reduce the number of memory accesses needed for the classification. Since the clock rate of an individual engine cannot catch up to that of the internal memory, multiple classification engines are used to increase the throughput. The implementations in two different FPGAs show that this architecture can reach a searching speed of 169 million packets per second (mpps) with synthesized ACL, FW and IPC rulesets. Further analysis reveals that compared to state of the art TCAM solutions, a power savings of up to 72% and an increase in throughput of up to 27% can be achieved.**

***Keywords- Packet classification, hardware accelerator, low-power, multi-engine.***

## I. INTRODUCTION

One of the major bottlenecks on a router line card is the process of packet classification. Packet classification involves matching information from a packet's header to a set of rules in order to determine the manner in which the packet should be processed by devices on the line card such as the network processor. The process of packet classification is an NP-hard problem which is further complicated by the fact that all packets entering a router must be processed at wire speed. The large number of services being provided by network providers makes this problem even more difficult as rulesets containing thousands of rules are needed to provide these services.

Implementing packet classification on devices such as programmable network processors is not a feasible option for core routers because line rates can reach a maximum throughput of 125 mpps. This is the case when you consider an OC-768 line rate and the back-to-back arrival of minimum-sized 40 byte packets. Using the network processor to implement packet classification would cause saturation even when using some of the best performing packet classification algorithms [1-9]. This would lead to packets being dropped and not allow the network processor time to carry out other important packet processing tasks.

For these reasons it can be seen that hardware approaches at packet classification are essential to prevent it becoming a bottleneck. The most common hardware approaches at packet classification include the use of Ternary Content Addressable Memory (TCAM). TCAM can help reach OC-768 line rates but is not always an ideal solution as it adds a large amount of power consumption to a routers already tight power budget. TCAMs use large amounts of power due to the fact they carry out a parallel compare of all stored rules in a single memory access in order to reduce lookup times. The structure of TCAM means it is not very efficient at storing rulesets. This is because a memory word's bits are stored in a 1, 0 or don't care state. This makes TCAM very efficient at storing fields which use longest prefix matching but poor at storing fields which use range matching. Range splitting must be performed to convert ranges into prefix formats. This further complicates the problem of power consumption as large amounts of memory are needed to store rulesets. There has been much research [10-12] into reducing the amount of power consumption and increasing the storage efficiency of rulesets but they still however remain a problem.

In [13] we present a packet classification hardware accelerator based on the HiCuts [1] and HyperCuts [2] packet classification algorithms. This hardware accelerator is capable of classifying at most 77 mpps when implemented on a FPGA. A modified version of this hardware accelerator was implemented in conjunction with an adaptive clocking unit in [14] to reduce power consumption by adapting the clock speed to meet fluctuations in traffic volume to a router's line card. In this paper we improve the trie traverser to reduce worst case number of clock cycles needed to classify a packet. Since the engines run slower than the maximum attainable clock speed of the internal memory, due to logic delay caused by ultra-wide memory word, we also investigate the possibility of using multiple packet classification engines in parallel with a shared memory interface in order to fully utilize the memory bandwidth provided by FPGA block RAM.

The architecture uses up to 4 packet classification engines working in parallel, when implemented on a Stratix 3 FPGA, capable of classifying rulesets containing up to 49,000 rules. These engines can be used to classify up to 169 mpps achieving OC-768 line speeds. The architecture uses up to 2 packet classification engines, when implemented on a Cyclone 3 FPGA, capable of classifying rulesets containing up to 24,000 rules at speeds of 65 mpps.

978-1-4244-4581-3/09/$25.00 ©2009 IEEE

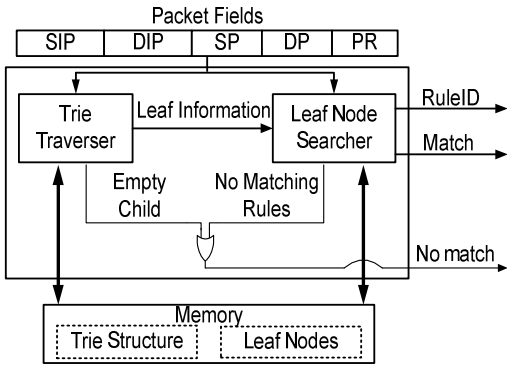


Figure 1. Packet classification engine architecture.

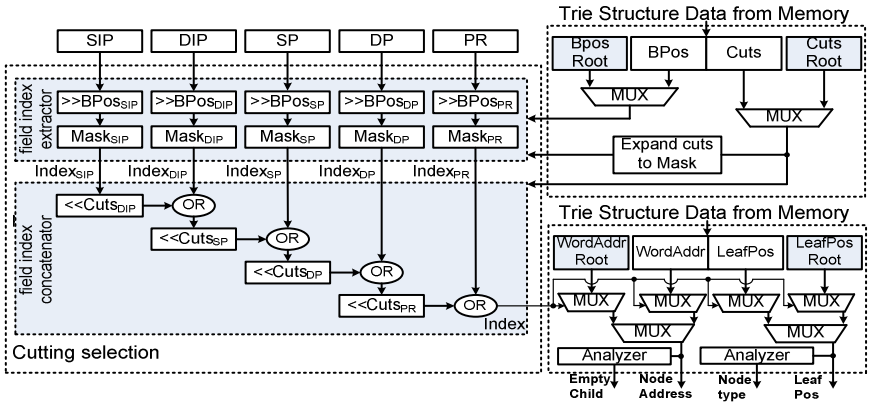


Figure 2. Trie traverser architecture.

The rest of the paper is organized as follows. Section II gives a detailed description of the decision tree based packet classification engine. In Section III the operation of multiple engines working in parallel is explained. Section IV and section V describe the hardware implementation on two types of FPGA and their performance analysis results with different engine configurations. Finally, Section VI concludes the paper.

## II. Architecture of Decision-Tree Based Packet Classification Engine

The design of our hardware accelerator follows the basic idea of decision-tree based packet classification algorithms, such as HiCuts [1], HyperCus [2], and modular scheme [8]. This type of method recursively cuts the hyperspace represented by the ruleset into smaller subspaces along some dimensions selected by several heuristics. This process continues in each subspace until either the number of rules it contains is smaller than a pre-defined value or it cannot be further divided. The recursive division can be represented by a trie (decision-tree) and the classifying a packet is to traverse down the trie until either an empty node or a leaf node is found. Within a leaf node, the rules can be searched in various ways. The simplest methods include linear searching, or using a small TCAM.

Fig. 1 shows the block diagram of our packet classification hardware accelerator. As is described above, two major steps in decision-tree based packet classification, is trie travsersing and leaf node searching, which are performed by the two modules shown in Fig. 1. When the five fields of an incoming packet have been extracted, they are used by the Trie Traverser to find the node corresponding to the subspace represented by their values. If the final child node is empty, the packet classification engine reports *Unmatch* and gets ready for the processing of next incoming packet. Otherwise, the address information of the leaf node is sent to the Leaf Node Searcher. In our implementation, the rules are compared with the five packet fields by hardware logic in parallel to speed up the searching process. If the rules spread in multiple words, the Leaf Node Searcher continues the comparisons until either a matching rule is found or the last rule in the leaf node is encountered.

In this implementation, we mainly targets FPGA devices, with their internal block memory being used for decision tree data structure storage. One advantage that FPGA devices can provide is the flexible memory word length, which does not have the constraints of external memory chips such as pin numbers and fixed data width. Almost every step of the packet classification procedure involves memory access, either trie structure or the rules in leaf node. This makes the design quite memory-centric, i.e. our major consideration is to reduce the number of memory accesses, without sacrificing too much memory utilization. On the other hand, ultra-wide memory word reduces the maximum possible working frequency of the packet classification engine. To increase the throughput, we also need to make a balance between the length of memory word and the clock rate. The implementation described in this paper has a word length of 7,704 bits for both the trie structure and leaf nodes, so that as much information as possible is fetched in one memory access with an acceptable working frequency. The rest of this section will explain the two major modules of the packet classification engine in details.

### A. Trie Traverser

Fig. 2 gives a detailed view of the Trie Traverser. Each internal node contains two types of information: the cutting scheme to be performed on the subspace represented by the node, and the addresses, i.e. pointers, of the child nodes generated from the cutting. When an internal node is fetched from the trie structured stored in memory, the cutting information is decoded and the packet fields are checked against the cutting scheme to decide the child node the packet fields belongs to. The index of this child node is used to select the corresponding pointer, whose content is further analyzed to see if it is empty, a leaf node, or another internal node.

Since each incoming packet goes through the root node, the root node information is stored in registers of the packet classification engine, to save one memory access for each classification. As can be seen in the right part of Fig. 2, some multiplexors are employed to do this switch between information coming from root node and internals nodes. In this implementation, the packet classification engine has only one memory interface, which means the Trie Traverser and Leaf Node Searcher cannot access the memory at the same time. By holding the root information in registers, it is possible traverse the root node with a newly arrived packet while searching the leaf node of a previous packet, making the two modules working in parallel to some extent.

Because the data width is quite large in this implementation, the memory organization is designed to be able to pack the whole internal node into one memory word. This has limited the maximum number of pointers that can be stored in the internal node. In this design, up to 512 subspaces are allowed to be divided for each node. The node pointer is 14-bit wide, and a 512-cutting scheme consumes 7,168 bits, with part of the remaining bits of one memory word used to encode the cutting scheme.

| Prefix Length | Bit[34:7] | Bit[6:3] | Bit[2:1] | Bit[0] |
|---|---|---|---|---|
| 32 | 32-Bit IP | | 00 | 0 |
| 31 | 32-Bit IP | | 01 | 0 |
| 30 | 32-Bit IP | | 10 | 0 |
| 29 | 32-Bit IP | | 11 | 0 |
| 28 | 28-Bit IP | 011100 | | 1 |
| 27 | 28-Bit IP | 011011 | | 1 |
| | | ⋮ | | |
| 1 | 28-Bit IP | 000001 | | 1 |
| 0 | 28-Bit IP | 000000 | | 1 |

Figure 3. Encoding scheme used for source and destination IP address.

The cutting information for each dimension consists of 2 pre-computed values: 1) *Cuts*, indicating how many subspaces are to be generated on this dimension. For the simplicity of hardware implementation, the number of subspaces is limited to be power of two. Therefore, the *Cuts* field is the result of $log_2$*(number of subspaces)*, where the maximum possible value is 9 for a 512-cutting scheme. Four bits are used as *Cuts* for each of the 5 dimensions. The value of *Cuts* is also the length of a bit-mask that picks up the bits of the packet field that are used to perform this cutting. As can be seen in Fig. 2, before the index of the child node is calculated, the *Cuts* information is extended to *Mask* for each dimension. 2) *BPos*, indicating the bit position where *Mask* should work on. In the calculation of child node index, *BPos* is the number of lower bits that need to be shifted out from the packet field, before the operation of ANDing with the *Mask* can be performed. The source and destination IP address will require a 5-bit *BPos* value as the 32-bit value will need to be shifted 0-31 places. Similarly the 16-bit port and destination numbers require a 4-bit *BPos* value while the 8-bit protocol number requires a 3-bit value.

As is shown in the left part of Fig. 2, the child node index calculation is performed in two steps: 1) extracting the sub-index for each dimension using the shift-right and mask operation, according to the *Cuts* and *BPos* information stored in the internal node; 2) concatenating these sub-indexes to form the final index. In this implementation, this concatenation operation is performed by left-shifting one sub-index by the length of the next one and then ORing them together, until the $index_{PR}$ is combined with the others.

Although FPGA device can have ultra-wide memory word, the size of the internal memory blocks limited the number of words that can be supported. For example, with a memory width of 7,704 bits, a Cyclone EP3C120 FPGA has 512 memory words and a Stratix EPSE260 FPGA doubles this number. Therefore, for FPGAs of similar types, 10 bits out of the 14-bit child node pointer is enough for the memory word address, as is labeled as *WordAddr* in Fig. 2. If all of the 10 bits are zero, it means this is an empty node and the *WordAddr* Analyzer will generate the *Empty Child Node* to indicate the end of classification for this packet. In other cases, this will be the word address of an internal node, or the address of the first word in a leaf node.

On the other hand, especially for small ruleset, it is often difficult for the leaf node to be large enough to fill into one ultra-wide memory word. To improve the memory utilization, the leaf node can start at positions other than the start of a memory word. For a leaf node, the memory word is divided into several chunks, with the *LeafPos* in the pointer indicating the index of the chunk where the leaf node starts. In this implementation, the 4-bit *LeafPos* can represent 15 different chunks for a leaf node and the value of 0b1111 indicates an internal node. If a leaf node is detected by the *LeafPos* Analyzer, it notifies the Leaf Node Searcher to start working on

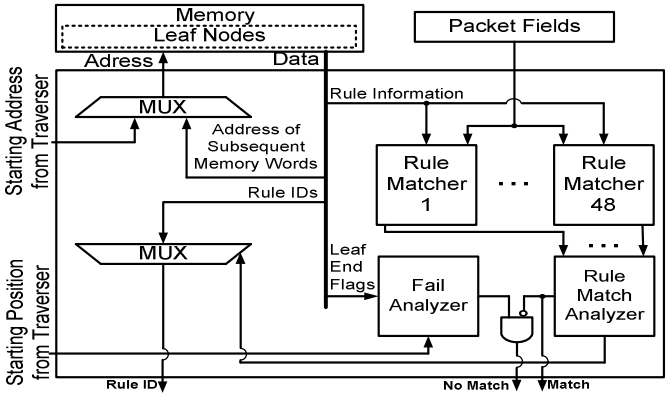


Figure 4. Leaf node searcher architecture.

the given address. Otherwise, the Trie Traverser begins to fetch the next internal node.

One thing that should be noted is that packing multiple leaf nodes into one memory word may result in extra cycles if the leaf node that otherwise can be stored in only one word is across multiple memory words. Therefore, the mapping of leaf nodes to memory word needs to be careful not to produce unnecessary memory accesses. Currently, we just stored the leaf nodes in the order of what they have been generated by the trie construction algorithm. If the next leaf node needs additional split when packing with previous nodes, we simply put it into a new memory word. However, some simple greedy algorithm can be used to improve the storage efficiency which does not obey the original order of leaf nodes.

### *B. Leaf Node Searcher*

Each rule requires 1 bit for determining if it is the last rule in a leaf node. If no rule has been matched by the time this flag is met set, the Leaf Node Searcher reports *No Matching Rules* and stops working for the current packet. Each rule also requires a 16-bit *Rule ID*. For the protocol field, 8 bits are used to store the protocol number and 1 bit for the mask. Both the source and destination ports require 16 bits for each of two boundaries of their ranges. The source and destination IP addresses use 35 bits to represent the prefixes. The lowest bit is used to indicate whether the prefix length is smaller than 28. If so, only 28 bits are needed to store the IP address, with rest 6 bits indicating the actual length. If not, all the 32 bits are used for IP with the rest two bits encoded to represent the four possible lengths. The detailed encoding scheme used for this implementation is shown in Fig. 3.

Each rule requires 160 bits, meaning that one 7,704-bit memory word can hold up to 48 rules. If a leaf node is too large to be stored in one memory word, the remaining bits are used to indicate the address information about the next memory word holding rules belonging to the leaf node.

The architecture of the Leaf Node Searcher is shown in Fig. 4. Since the address of the leaf node can come from Trie Traverser for the first memory word, or from the loaded memory for the subsequent word, a multiplexer is needed to correctly load from the memory. The Leaf Node Searcher consists of 48 comparator logic blocks which work in parallel to search a matching rule. The logic for the comparators is trivial and is not described in this paper. The matching results of the comparators are fed into an analyzer, which will report the first matching rule and select its Rule ID for output. Note that rules from other leaf nodes never match the incoming packet, so their comparison result will not affect the final result.

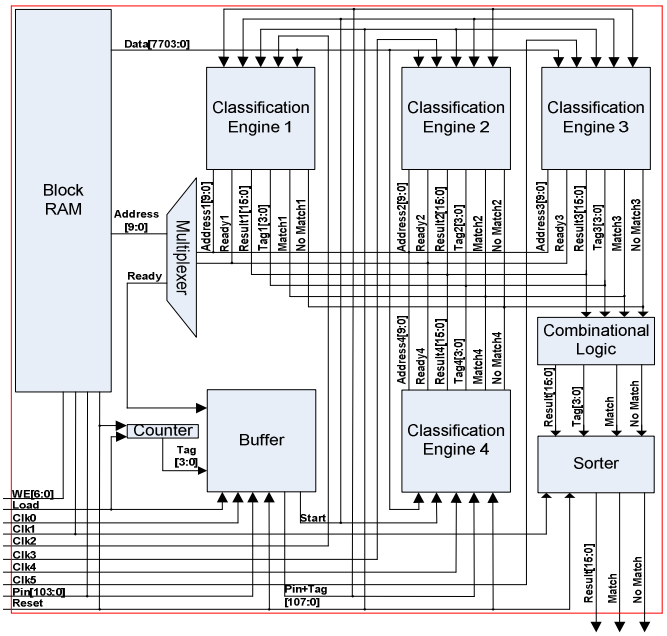


Figure 5. Hardware accelerator architecture.

However, the final rule flags of other leaf nodes do affect the judgement when no matching rules are found in the current memory word. As shown in Fig. 4, a fail analyzer is responsible for generating No Matching Rules signal or indicating the fetching of another memory word. It takes both the final rule flag and the starting position of the leaf node as input. The *No Matching Rules* signal is asserted if there is no comparator reporting a matching rule and a final rule flag greater than or equal to the starting position is detected. If neither the *No Matching Rules* signal nor the *Match* signal is asserted, that means the leaf node has not been fully searched and the memory address for the subsequent rules in the leaf node stored in the same memory word is used.

## III. Multi-Engine Packet Classification Hardware Accelerator

A diagram of the hardware accelerator using 4 classification engines is shown in Fig. 5. The hardware accelerator has a 108-bit buffer used to store the incoming 104-bits used to classify each packet and a 4-bit packet tag. The packet tag is used to make sure the packet classification results outputted by the hardware accelerator are given in the same sequence as the packets are inputted. Every time a packet appears at the input a *load* pin will be asserted. This *load* pin will increment a 4-bit counter used to create a packet tag and the address of the buffer where the packet will be saved. The 104-bits from the packet header and the 4-bits from the counter will be saved together in the buffer. The buffer shown in Fig. 5 can store up to 256 packets and their tags at speeds in excess of 125 MHz, allowing it to operate at OC-768 line rates. The hardware accelerator multiplexes the *ready* signal from the packet classification engines together. The *ready* signal indicates a packet classification engine is ready to classify a packet. A *ready* signal being asserted will load a packet and its tag from the buffer. It will also cause the buffer's load address to be incremented.

In order to simplify the design, the block RAM address issued from each packet classification engine is multiplexed together instead of using a command buffer as in many other designs. Suppose the engine issues one memory access requirement on each cycle. Then the block RAM runs at a speed that equals to the sum of each engine. The frequency of each engine has a different phase in order to make sure that on every clock cycle of the block RAM, it receives an access requirement.

TABLE I
FPGA Resource Utilization

| Device | EP3SE260F115C3 | | | | EP3C120F484C7 | |
|---|---|---|---|---|---|---|
| Engines | 1 | 2 | 3 | 4 | 1 | 2 |
| M9Ks | 859 | 859 | 859 | 859 | 431 | 431 |
| Logic Elements | 24,992 | 58,100 | 90,970 | 121,797 | 20,633 | 45,244 |

Each packet classification engine will assert a *match* or *no match* signal every time it has finished classifying a packet. This *match* and *no match* signal along with the corresponding matching rule number and packet tag from each of the packet classification engines are multiplexed together. They are then inputted into a logic block used to sort out the matching rule numbers from the engines, so that they are outputted from the hardware accelerator in the correct sequences. The sorter logic block consists of a chain of 16 registers and 15 multiplexers in series. Control logic will register a matching rule number or blank rule number to a register when a *match* or no *match signal* is asserted. The register selected will depend on the packet tag number. The rule number will be registered to the output register if it is next in the sequence of packet results to be outputted and stored if not. All stored rules will be shifted towards the output register each time a rule appears which is due to be outputted. This process is hidden, with the hardware accelerator outputting the result of classified packets on a first come, first served basis. The architecture differs from that shown in Fig. 5 when only 1 classification engine is used. The sorter logic block and multiplexers used for multiplexing the output signals of the classification engines are not needed.

## IV. FPGA Implementation

We implement the multi-engine packet classifier using VHDL on a Stratix EP3SE260F115C3 and a Cyclone EP3C120F484C7 FPGA. The former has 864 M9K block RAMs, 254,400 Logic Elements (LEs) and uses 1.1 Volts. The latter has 432 M9Ks and 119,088 LEs with 1.2 Volts. Both of them are built on Taiwan Semiconductor Manufacturing Company's (TSMC's) 65-nm process technology. The architectures were synthesized using Altera's Quartus 2 design software. Table I shows the memory and logic resources needed to implement the hardware accelerator. Note that it is a lack of available interconnect and not logic elements that prevent further engines being added.

## V. Performance Analysis

### A. Memory usage and worst-case number of clock cycles

The results in Fig. 6 show the memory needed to save search structures built for synthetic rulesets generated using ClassBench [15]. To build these search structures the modified HyperCuts algorithm was used. The actual number of rules for the ACL1, IPC1 and FW1 rulesets containing 25,000 rules is 24,920, 24,274 and 23,087, respectively. This is due to the way Classbench generates rulesets. The figure also shows the worst-case number of clock cycles needed to classify a packet. The hardware accelerator can save search structures for rulesets containing up to 49,000 rules, when implemented on a Stratix EP3SE260F115C3 FPGA, and rulesets containing up

to 24,000 rules when implemented on a Cyclone EP3C120F484C7 FPGA.

The search structures built using the ACL1, IPC1 rulesets for both the Cyclone and Stratix implementation show similar performance results. This is because memory consumption is not a major problem for these rulesets as they don't contain many rules with wildcard fields. For the search structure built using the FW1 rulesets it can be seen that the Stratix implementation shows better performance than the Cyclone implementation. This is because memory consumption is a problem due to the many rules containing wildcard fields. The FW1 ruleset for example with 15,000 rules needs, at worst 9 clock cycles to classify a packet when 3,944,448 bits of memory are available for the search structure using the Cyclone. This figure is reduced to 4 clock cycles when the amount of memory available is doubled using a Stratix.

### *B. Power consumption*

The power consumed by the hardware accelerator is shown in Fig. 7 and 8, with the results for the hardware accelerator generated using a search structure requiring at worst 2 clock cycles to classify a packet. Post place and route simulations were carried out using Quartus 2 PowerPlay Power Analyzer Tool with VCD files generated by ModelSim. These results were compared to the power consumed by the state of the art Cypress Ayama 10000 Network Search Engine [16], which uses similar amounts of memory. The hardware accelerator implemented on the Cyclone 3 FPGA with 3,944,448 bits of memory is compared to the Cypress Ayama 10128 search engine with 4,608,000 bits of TCAM. The Stratix 3 implementation of the hardware accelerator with 7,888,896 bits of memory available is compared to the Cypress Ayama 10256 search engine with 9,216,000 bits of TCAM.

Looking at Fig. 7 it can be seen that the TCAM has a maximum throughput of 133 mpps, while the hardware accelerator implemented on the Cyclone 3 FPGA with 2 packet classification engines has a maximum throughput of 65 mpps when running at 65 MHz. This throughput decreases to 36 mpps when 1 engine is used while running the hardware accelerator at 36 MHz. These levels of throughput are more than enough to cope with line rates up to OC-768 as we explain later. At these speeds the hardware accelerator shows an energy saving of 66.38% when classifying 65 mpps with 2 engines, and 72.24% when classifying 36 mpps with 1 engine, when compared to the TCAM running at the same speed.

Fig. 8 shows that for the Stratix 3 implementation of the hardware accelerator a maximum throughput of 169 mpps is obtainable when 4 packet classification engines are used. This compares well to the TCAM which has a maximum throughput of 133 mpps. The maximum throughput for the hardware accelerator drops to 136 mpps when 3 packet classification engines are used, 108 mpps when 2 engines are used and 68 mpps when 1 engine is used. The hardware accelerator shows a power saving of 46.35% when classifying 133 mpps with 4 engines, 49.85% when classifying 133 mpps with 3 engines. 54.01% when classifying 108 mpps with 2 engines and 53.85% when classifying 68 mpps with 1 engine compared to the TCAM running at the same speeds.

### *C. Performance on Real-Life Packet Traces*

In order to test the packet hardware accelerator synthetic OC-48, OC-192 and OC-768 packet traces were created by aggregating Abilene, CENIC, and SCO4 backbone packet

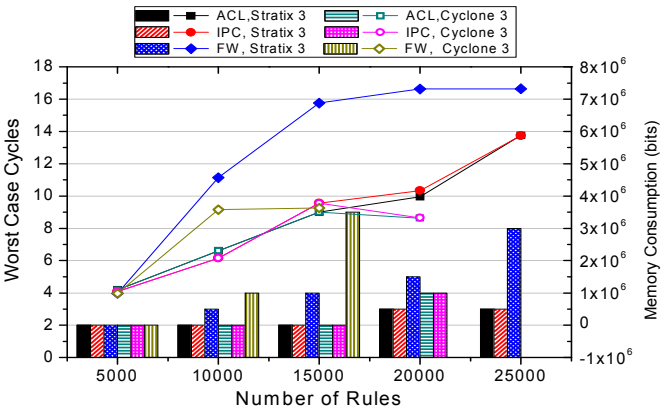


Figure 6. Memory usage(lines) and worst case number of clock cycles(bars).

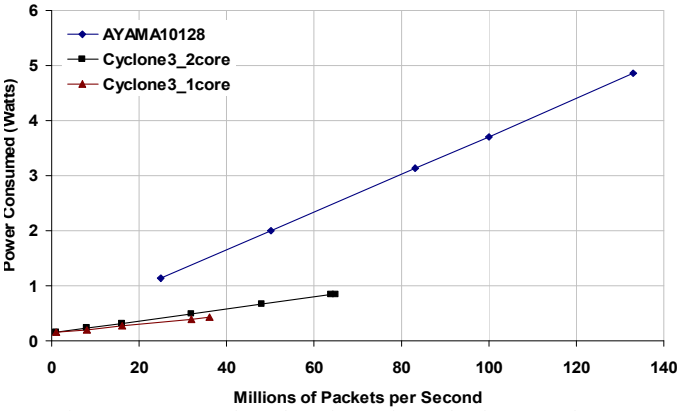


Figure 7. Power vs. throughput for Cyclone 3 hardware accelerator.

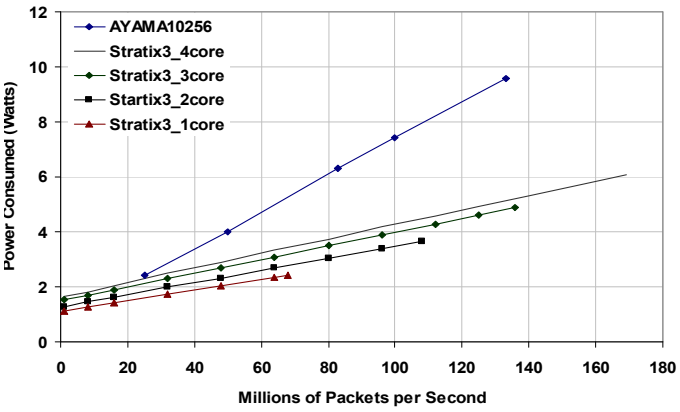


Figure 8. Power vs. throughput for Stratix 3 hardware accelerator.

traces until peak line rates of 2.5, 10 and 40 Gb/s were reached. These traces were obtained from NLANR [17]. The OC-48 and OC-192 traces were looked at over a 6000 second period while the OC-768 trace was looked at for 2000 seconds. The peak numbers of packets per second for the traces are 143,768 p/s for the OC-48 trace, 661,526 p/s for the OC-192 trace and 3,302,488 p/s for the OC-768 trace.

The timestamp from these traces were spliced to packet headers created using ClassBench for the ACL1, FW1 and IPC1 rulesets. A cycle accurate simulator for the multi-engine packet classification hardware accelerator was developed in C code to analyze these traces. Table II shows the maximum number of packets which were buffered when using these traces. The hardware accelerator was tested with 1, 2, 3 and 4 classification engines for the Startix 3 implementation and with 1 and 2 engines for the Cyclone 3 implementation. For all traces the memory was simulated running at a constant speed while the number of classification engines accessing it was

TABLE II
MAXIMUM BUFFER USAGE

| Device | | Stratix 3 | | | | Cyclone 3 | | | Stratix 3 | | | | Cyclone 3 | | | Stratix 3 | | | | Cyclone 3 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Speed | | 2 MHz | | | | | | | 8 MHz | | | | | | | 32 MHz | | | | | |
| Engines | | 1 | 2 | 3 | 4 | 1 | 2 | | 1 | 2 | 3 | 4 | 1 | 2 | | 1 | 2 | 3 | 4 | 1 | 2 |
| ACL5000 | OC-48 | 17 | 17 | 17 | 17 | 17 | 17 | OC-192 | 28 | 27 | 29 | 29 | 28 | 27 | OC-768 | 18 | 19 | 20 | 21 | 18 | 19 |
| ACL10000 | | 17 | 17 | 17 | 17 | 17 | 17 | | 28 | 27 | 29 | 29 | 28 | 27 | | 18 | 19 | 20 | 21 | 18 | 19 |
| ACL15000 | | 17 | 17 | 17 | 17 | 17 | 17 | | 28 | 27 | 29 | 29 | 28 | 27 | | 18 | 19 | 20 | 21 | 18 | 19 |
| ACL20000 | | 17 | 17 | 17 | 17 | 32 | 32 | | 28 | 27 | 29 | 29 | 44 | 44 | | 18 | 19 | 20 | 21 | 44 | 43 |
| ACL25000 | | 17 | 17 | 17 | 17 | | | | 28 | 27 | 29 | 29 | | | | 18 | 19 | 20 | 21 | | |
| FW5000 | | 17 | 17 | 17 | 17 | 17 | 17 | | 28 | 27 | 29 | 29 | 28 | 27 | | 18 | 19 | 20 | 21 | 18 | 19 |
| FW10000 | | 20 | 20 | 19 | 20 | 24 | 24 | | 36 | 36 | 36 | 35 | 42 | 42 | | 24 | 25 | 26 | 27 | 35 | 35 |
| FW15000 | | 20 | 21 | 21 | 21 | 49 | 49 | | 40 | 39 | 40 | 39 | 61 | 60 | | 33 | 33 | 33 | 33 | 57 | 57 |
| FW20000 | | 33 | 33 | 33 | 33 | | | | 50 | 49 | 50 | 50 | | | | 45 | 45 | 44 | 44 | | |
| FW25000 | | 46 | 46 | 46 | 46 | | | | 57 | 57 | 55 | 55 | | | | 53 | 53 | 53 | 53 | | |
| IPC5000 | | 17 | 17 | 17 | 17 | 17 | 17 | | 28 | 27 | 29 | 29 | 28 | 27 | | 18 | 19 | 20 | 21 | 18 | 19 |
| IPC10000 | | 17 | 17 | 17 | 17 | 17 | 17 | | 28 | 27 | 29 | 29 | 28 | 27 | | 18 | 19 | 20 | 21 | 18 | 19 |
| IPC15000 | | 17 | 17 | 17 | 17 | 17 | 17 | | 28 | 27 | 29 | 29 | 28 | 27 | | 18 | 19 | 20 | 21 | 18 | 19 |
| IPC20000 | | 17 | 17 | 17 | 17 | 18 | 18 | | 28 | 27 | 29 | 29 | 33 | 32 | | 18 | 19 | 20 | 21 | 27 | 28 |
| IPC25000 | | 17 | 17 | 17 | 17 | | | | 28 | 27 | 29 | 29 | | | | 18 | 19 | 20 | 21 | | |

varied in order to record the maximum number of packets buffered for the different number of classification engines. For the OC-48 the memory was run at a speed of 2 MHz. At this speed the hardware accelerator was easily able to cope with the line rate. The Stratix 3 implementation recorded a peak of 46 buffered packets while the Cyclone 3 implementation recorded a peak of 49 packets. For the OC-192 traces the memory was run at 8 MHz with the Startix 3 implementation recording a peak of 57 packets and the Cyclone 3 implementation recording a peak of 55 packets buffered. Finally for the OC-768 traces the memory was run at 32 MHz, with a peak of 53 packets buffered by the Stratix 3 implementation, and the Cyclone 3 implementation buffering a peak of 57 packets. These results show that the maximum frequency of 169 MHz for the Stratix 3 block RAM and 65 MHz for the Cyclone 3 block RAM is more than enough processing power to cope with line speeds of up to OC-768.

## VI. CONCLUSION

In this paper we have presented an architecture for a packet classification hardware accelerator which uses multiple engines to relieve the bottle neck of packet classification on a router's line card. It can classify up to 169 mpps when implemented on Stratix 3 FPGA, which means it can easily meet OC-768 line rates. The architecture presented in this paper shows an increase in throughput of up to 36 mpps when compared to state of the art TCAM solutions. This is possible due to the ultra-wide memory word which reduces the number of memory accesses, and the higher clock speed obtainable by internal memory of FPGA. The hardware accelerator presented can also reduce power consumption by up to 72%. This power reduction is achieved by only comparing 48 rules at a time rather than a compare of all rules as is the case with the TCAM solution.

## ACKNOWLEDGMENT

This work was funded by the Irish Research Council for Science, Engineering and Technology, funded by the National Development Plan; NSFC (60625201, 60873250); and the Cultivation Fund of the Key Scientific and Technical Innovation Project, MOE, China (705003).

## REFERENCES

[1] P. Gupta and N. McKeown, "Packet classification using hierarchical intelligent cuttings," *IEEE Micro*, vol.20, no. 1, pp. 34-41, 2000.

[2] S. Singh, F. Baboescu, G. Varghese and J. Wang, "Packet Classification Using Multidimensional Cutting" *in ACM SIGCOMM*, 2003, pp.213-214

[3] P. Gupta and N. McKeown, "Packet classification on multiple fields," *in ACM SIGCOMM 1999*, pp.147-160

[4] F. Baboescu and G. Varghese, "Scalable packet classification," *IEEE/ACM Trans. Netw.*, vol. 13, no. 1 pp. 2-14, 2005.

[5] F. Baboescu, S. Singh, and G. Varghese, "Packet classification for core routers: Is there an alternative to CAMs?" *in IEEE INFOCOM*, 2003, pp. 53-63.

[6] V. Srinivasan, S. Suri, and G. Varghese, "Packet Classification using Tuple Space Search" *in ACM SIGCOMM 1999*, pp. 135-146.

[7] P. Gupta and N. McKeown, "Algorithms for packet classification," *IEEE Network Mag.*, vol. 15, no. 2, pp.24-32, 2001.

[8] T. Woo, "A modular approach to packet classification: algorithms and results," *in IEEE INFOCOM*, Mar. 2000, pp. 1213-1222.

[9] P. C. Wang, C. T. Chan, C. L. Lee and H. Y. Chang "Scalable Packet Classification for Enabling Internet Differentiated Services" *IEEE Trans. on Multimedia*, vol. 8, no. 6, pp. 1239-1249, 2006.

[10] E. Spitznagel, D. Taylor, and J. Turner, "Packet Classification Using Extended TCAMs," *Proc. 11th Int'l Conf. Network Protocol (ICNP '03),* 2003.

[11] K. Zheng, H. Che, Z. Wang, B. Liu, X. Zhang, "DPPC-RE: TCAM-Based Distributed Parallel Packet Classification with Range Encoding," *IEEE Transactions on Computers*, vol. 55, no. 8, pp. 947-961, Aug., 2006.

[12] D. Pao, Y Keung Li, P Zhou, "An encoding scheme for TCAM-based packet classification" *Advanced Communication Technology*, Feb. 2006.

[13] A. Kennedy, X. Wang. B. Liu "Energy efficient packet classification hardware accelerator" *Proceedings of the 22nd IEEE International Parallel and Distributed Processing Symposium,* IPDPS Miami 2008.

[14] A. Kennedy, X. Wang, Z. Liu, B. Liu. "Low Power Architectures for High Speed Packet Classification" *Proc of the 2008 ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, 2008, California, 6-7 Nov.

[15] D. Hoffman and P. Strooper, "Classbench: A Framework for Automated Class Testing," *Software Practice and Experience*, vol. 27, no. 5, pp. 573-597, May 1997.

[16] Cypress Ayama 10000 Network Search Engine, http://download.cypress.com.edgesuite.net/design_resources/datasheets/contents/cynse10256_8.pdf

[17] Passive Measurement and Analysis Project, National Laboratory for Applied Network Research. http://pma.nlanr.net