Low Power Architecture for High Speed Packet Classification

Alan Kennedy
School of Electronic Engineering
Dublin City University
Dublin 9, Ireland
alan.kennedy@eeng.dcu.ie

Xiaojun Wang
School of Electronic Engineering
Dublin City University
Dublin 9, Ireland
xiaojun.wang@dcu.ie

Zhen Liu
School of Electronic Engineering
Dublin City University
Dublin 9, Ireland
liuzhen@eeng.dcu.ie

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

ABSTRACT

Today's routers need to perform packet classification at wire speed in order to provide critical services such as traffic billing, priority routing and blocking unwanted Internet traffic. With ever-increasing ruleset size and line speed, the task of implementing wire speed packet classification with reduced power consumption remains difficult. Software approaches are unable to classify packets at wire speed as line rates reach OC-768, while state of the art hardware approaches such as TCAM still consume large amounts of power.

This paper presents a low power architecture for a high speed packet classifier which can meet OC-768 line rate. The architecture consists of an adaptive clocking unit which dynamically changes the clock speed of an energy efficient packet classifier to match fluctuations in traffic on a router line card. It achieves this with the help of a scheme developed to keep clock frequencies at the lowest speed capable of servicing the line card while reducing frequency switches. The low power architecture has been tested on OC-48, OC-192 and OC-768 packet traces created from real life network traces obtained from NLANR while classifying packets using synthetic rulesets containing up to 25,000 rules. Simulation results of our classifier implemented on a Cyclone 3 and Stratix 3 FPGA, and as an ASIC show that power savings of between 17-88% can be achieved, using our adaptive clocking unit rather than a fixed clock speed.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General Security and protection (e.g., firewalls); C.2.6 [**Internetworking**]: Routers

General Terms

Measurement, Performance, Experimentation, Security.

Keywords

Packet Classification, Hardware Accelerator, Frequency Scaling, Energy Efficient.

ANCS'08, November 6–7, 2008, San Jose, CA, USA.

1. INTRODUCTION

Packet classification is the process of mapping a packet to one of a finite set of flows or categories using information from the packet header. The packet header is matched using longest prefix matching for the source and destination IP addresses, range matching for the source and destination ports, and exact or wildcard matching for the protocol number. Packets belonging to the same flow match a pre-defined rule and are processed in the same way by a router line card. The classifier will select the rule with the highest priority in the case where multiple rules match. Packet classification is used by networking devices to carry out advanced Internet services like network security, sophisticated traffic billing, giving priority to VoIP and IPTV packets, rate limiting, load balancing and resource reservation.

The increasing number of services that the router needs to provide means that the number of rules used to separate incoming packets into appropriate flows has grown from hundreds to thousands of rules. This growth in ruleset size has further complicated the problem of packet classification. The problem of packet classification has been looked at in detail [1-9]. Implementing packet classification algorithms in software is not feasible when trying to achieve high speed packet classification [10]. High throughput algorithms such as RFC [1] are unable to reach OC-768 or even OC-192 line rates when run on devices such as general purpose processors for even relatively small sized rulesets. Packet classification algorithms tailored towards high throughput also tends to suffer from large memory consumption for rulesets containing thousands of rules, making them more suitable for slower DRAM rather than faster SRAM.

These approaches at packet classification seldom consider power consumption, which is an equally important factor. Key components on a router line card such as the Intel IXP2800 network processor can consume up to 30 Watts. Each line card on a router typically contains two network processors for ingress and egress processing and a router can contain multiple line cards. Current hardware methods for high speed packet classification such as Ternary Content Addressable Memory (TCAM) can meet OC-768 line rate but tend to use large amounts of power. State of the art low power packet classification devices such as the Cypress Ayama 10000 Network Search Engine [11] can use up to 19.14 Watts. This is due to the fact that TCAM carries out parallel comparisons on all the stored rules in one clock cycle. TCAM also has a poor storage density with one bit requiring 10-12 transistors, compared to 4-6 transistors for more power efficient SRAM. Another drawback of TCAM is that it can suffer from poor rule storage efficiency when classifying rules that use range

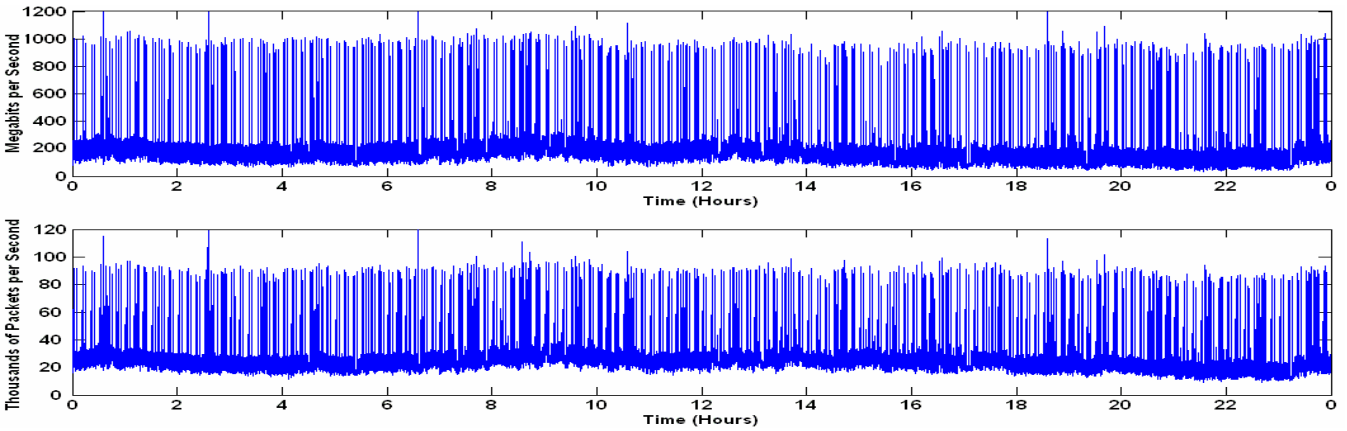

Figure 1. Throughput for a sample 24-hour trace taken from CENIC backbone link

matching. This is because rules that use range matching may require more than one memory slot in order to store a rule. A search engine implemented using TCAM also has the drawback of requiring multiple chips including a host ASIC and the corresponding SRAMs. Improving TCAM power efficiency and the storage efficiency of rulesets has been well investigated [12-14] but there is still large room for improvement.

In this paper, we propose a low power architecture for a high speed packet classifier which could be used as an on-chip hardware accelerator for a network processor or as an external chip. This architecture uses an adaptive clocking unit to exploit the fluctuation in Internet traffic by reducing the clock frequency during times of low traffic and increasing the clock frequency at times of high traffic. In order to make the decision of frequency scaling, the fields of a packet header used for classification are extracted into a buffer upon its arrival and the queue length is monitored. The hardware accelerator implements a modified version of the HiCuts and HyperCuts packet classification algorithms. Instead of comparing a large number of rules simultaneously (as is the case with TCAM), the algorithms divide the hyperspace of the ruleset heuristically into multiple groups so that each subset contains only a small number of rules that are suitable for linear search, reducing the unnecessary comparisons and thus the power consumption. The hardware accelerator utilizes the flexibility of a FPGA's block RAM by using SRAM with long word line to reduce the number of clock cycles needed to perform a linear search on the selected rules. We present performance results for our low power architecture implemented on a Cyclone 3 FPGA with 3,971,072 bits of memory capable of storing up to 24,000 rules, and on a Stratix 3 FPGA and as an ASIC, both of which possess 7,915,520 bits of memory capable of storing up to 49,000 rules.

The rest of the paper is organized as follows: In section 2 we explain our motivation behind using frequency scaling to reduce the dynamic power consumption and show our analysis of real life network traces. Section 3 introduces our scheme for frequency scaling. In section 4, we represent the HiCut and HyperCut packet classification algorithms along with the changes made to make them better suited for hardware acceleration. The architecture for our low power high speed packet classifier is described in section 5. Section 6 presents the power analysis results of our low power architecture being used to classify packets at different line rates using different sized and shaped rulesets. Section 7 concludes.

2. MOTIVATION

The Internet backbone is made up of a large collection of interconnected commercial and non-commercial high speed data links. These links are connected by edge and core routers. In the past OC-48 connections were used as the backbones by many regional Internet service providers. This corresponds to link speeds of 2.5 Gb/s which means it is possible to transmit a maximum of 7.8125 Million packets per second (Mpps) when you consider the back-to-back arrival of minimum-sized 40-byte packets. Currently the common commercial network connection speed is OC-192, which can transmit 10 Gb/s with a maximum throughput of 31.25 Mpps. With companies like AT&T already using OC-768 link speeds, it is envisaged that in the near future these OC-768 connections will become more commonly available, transmitting 40 Gb/s with a maximum throughput of 125 Mpps.

During peak times such as office hours a router line card may be kept busier than at other times such as night or public holidays. At a micro level traffic volume can also fluctuate from second to second with large peaks and troughs. We analyzed the characteristics of real life OC-48 and OC-192 traffic traces stored in the NLANR database [15]. We looked at throughput both in terms of bits and packets per second. Packet classifiers are more interested in throughput in terms of packets per second rather than bits per second. This is because packet classifiers only examine a packet header and not its payload. Figure 1 shows a 24-hour recording taken from the Corporation for Education Network Initiatives in California (CENIC) HPR backbone link [16]. Its characteristics are typical of all the backbone traces we analyzed, with the average traffic load varying from hour to hour with many short bursts in throughput. It can be seen that these short bursts cause the throughput to fluctuate wildly from second to second both in terms of bits and packets per second. The traces show that even during sharp bursts in throughput the 10 Gigabit CENIC backbone link peaks at 121,801 packets per second and never reaches its theoretical highest throughput of 32 Mpps. This is due to the fact that the back-to-back arrival of 40 byte packets is rare. A breakdown of the packet length distribution can be seen in Table 1.

Table 1. Statistics from CENIC backbone trace

Number of Packets	Average Packet Length	Packet Length Distribution		
		0-200	201 -1400	1401-1600
2,607,169,713	975 bytes	33.56 %	7.03 %	59.41 %

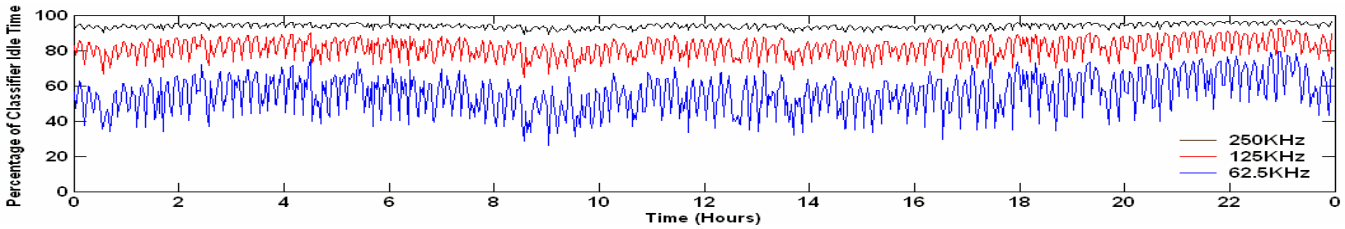

Figure 2. Percentage of time classifier spends idle when classifying packets from the CENIC trace at different frequencies

There have been many ideas proposed to reduce power consumption to a router line card by exploiting the fluctuation in traffic volume. In [17] clock gating is used to turn off the clock of unneeded processing engines of multicore network processors to save dynamic power at times when there is a low traffic workload. In [18] the more aggressive approach of turning off these processing engines is used to reduce both dynamic and static power consumption. Dynamic voltage scaling is used in [19] to reduce the power consumption of the processing engines on a network processor. With state of art devices such as the Cypress Ayama 10000 Network Search Engine using nearly as much power as that of a programmable network processor, our goal is to reduce the classifiers power consumption by means of exploiting the fluctuation in traffic volume to adjust the frequency dynamically. We do not use clock gating because a router line card typically does not contain more than one packet classifier. This means that the packet classifier must be continuously available for use to prevent packets from being dropped or unacceptable processing latencies that might be caused by turning off its clock. Since our experiments use FPGA which is harder for dynamic voltage scaling to be implemented on and may need external circuitry to control the voltage level [20], we will leave this work for the future.

We developed a cycle accurate simulator for our low power architecture for high speed packet classification in C code and used it to analyze traffic traces. Figure 2 shows the percentage of time our packet classifier spent in an idle state when classifying packets from the CENIC trace at different fixed clock speeds. For this experiment we replaced the source and destination IP addresses, the source and destination ports and the protocol number with header information generated using ClassBench [21] to match a synthetic ruleset. Our classifier is able to classify a packet on each clock cycle for the synthetic ruleset used. It can be seen that for higher clock speeds the packet classifier spends a large amount of time in an idle state. A large percentage of idle time means a large amount of unnecessary dynamic power is being used due to the unnecessary switching of logic and memory elements. As the packet classifier's clock speed is reduced to meet the average throughput of the router, it can be seen that the percentage of idle time decreases, meaning a reduction in the amount of unnecessary dynamic power used by the classifier. Running the packet classifier at a fixed clock speed close to the average throughput means a large buffer will be required to accommodate large bursts in throughput. A large buffer would cancel out the dynamic power saved by the packet classifier and cause an unacceptable latency in the amount of time it takes to classify a packet.

It was with these facts in mind that we decided to design an adaptive clocking unit which would dynamically scale the frequency of a packet classifier so that it matches fluctuations in traffic volume. It is possible to reduce the classifiers dynamic power consumption by running it at low speeds when traffic volume is low. It is also possible to reduce the buffer size and therefore its power consumption, by allowing the classifier to respond to bursts of packets, through increasing its clock frequency in order to keep the buffer clear.

3. ADAPTIVE CLOCKING ARCHITECTURE

Our adaptive clocking unit uses dual port SRAM to buffer information from the packet headers. This information includes the source and destination IP addresses, the source and destination port numbers and the protocol number, which are read in at a speed of 128 MHz. This speed is selected to avoid packets being dropped when the arrival of back-to-back 40-byte packets occur at OC-768 line speeds resulting in up to 125 Mpps as mentioned before. The number of packets stored in the buffer is calculated by monitoring the difference between the read and write addresses of the buffer. This difference is used as a trigger to determine which clock frequency the packet classification hardware accelerator should be run at. The adaptive clocking unit is designed to run a packet classification hardware accelerator at up to N different frequencies and in our experiment we have $N=10$. Each frequency is generated using a separate Phase Lock Loop (PLL) to eliminate the need of PLL frequency changing which requires some time to finish. Dedicated clock switching logic in the FPGA is used to prevent clock glitches when switching between frequencies. Before switching to another frequency, we need to put the packet classifier into an idle state to prevent problems that may be caused by glitches.

The selection of frequencies that the packet classifier is allowed to run at can use different schemes. In our experiments, we choose the following equation:

$$f_i = Fmax/2^{N-i-1},\ i=0, \ldots, N-1 \qquad (1)$$

Where $Fmax$ is the maximum frequency that the classifier can run. For our packet classifier, it was found that 32 MHz is fast enough to deal with the worst case bursts of packets for OC-768 line speeds when using rulesets containing over 20,000 rules with real life traces. This means that $Fmax=32\ MHz$. In our adaptive clocking unit, each possible clock frequency corresponds a different state. Table 2 shows the clock frequencies associated with each of these states. The entering and exiting of each state is triggered by the number of packets stored in the buffer. All states

Table 2. Clock speed associated with each state

State	S_0	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9
Speed MHz	f_0=0.0625	f_1=0.125	f_2=0.25	f_3=0.5	f_4=1	f_5=2	f_6=4	f_7=8	f_8=16	f_9=32

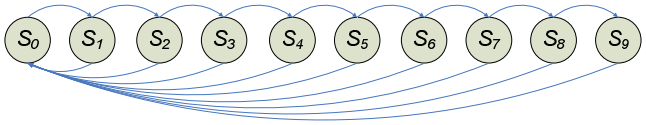

Figure 3. Switching sequences with all states used

apart from state S_{N-1} has a threshold for determining how many packets can be stored in the buffer before the next higher frequency is used. This threshold is variable with the number of packets stored in the buffer distributed among the N states with each state having a width W_i. The width of each state W_i can be any number between zero and M (total number of packets the buffer can store) as long as the following equation is satisfied:

$$M = \sum_{i=0}^{N-1} W_i \qquad (2)$$

The threshold for determining when a state is exited and the next higher state entered is saved in a register in the adaptive clocking unit and can be changed at any time. The threshold for each state is calculated using the following equation:

$$T_i = \sum_{j=0}^{i} W_j, \; i=0, \ldots, N-2 \qquad (3)$$

The output clock frequency of the adaptive clocking unit always starts at the lowest-used frequency of the hardware accelerator and only changes to the frequency of the next higher-used state if the number of packets stored in the buffer exceeds its threshold. There are two conditions for leaving the subsequent states and thus changing the output clock frequency. The first of these conditions is that the threshold T_i for the current state S_i is exceeded with the output clock frequency scaling up to the next higher-used frequency. The second condition is that the number of packets stored in the buffer reaches zero meaning the output clock frequency scales down to the lowest-used frequency. This means that the number of buffer slots that the current state can occupy before a frequency change is equal to the sum of the buffer slots occupied by the previous states plus the number of slots assigned to the current state itself. This is done to allow larger fluctuations in the number of packets stored in the buffer without unnecessary frequency drops. It also keeps the latency time of processing a packet to a minimum, by trying to clear the buffer before reducing the clock frequency. The clock frequency of the packet classification hardware accelerator remains fixed if all buffer slots are occupied by one state.

In the example shown in Figure 3 the buffers slots are distributed equally among all states. The output clock frequency to the packet classification hardware accelerator will start at the frequency of the lowest-used state f_0. If the threshold for this state T_0 is exceeded (i.e. the buffer slots assigned to state S_0 have been filled) then the next higher-used state S_1 will be entered and the clock frequency will change to f_1. The output clock frequency will remain at f_1 until the number of packets stored in the buffer is reduced to zero, returning the output clock frequency to f_0, or the threshold T_1 is exceeded in which case the output clock frequency changes to f_2. The same is true for all subsequent states. The output clock frequency will remain at f_2 until either all packets in the buffer are cleared returning the output clock frequency to f_0, or the maximum threshold T_2 is exceeded, meaning state S_3 is entered and the output clock frequency changes to f_3.

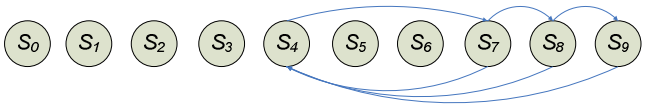

Figure 4. Switching sequences with selected states used

Figure 4 shows an example where only states S_4, S_7, S_8 and S_9 are used. In this case the output clock frequency to the packet classifier will start at f_4. It will stay at f_4 until the threshold T_4 is exceeded, increasing the clock frequency to f_7. The output clock frequency will stay at f_7 until all packets in the buffer have been cleared, returning the output frequency to f_4, or the threshold T_7 is exceeded, increasing the output frequency to f_8. The same procedure is followed for states S_8 and S_9.

4. HARDWARE ACCELERATOR

The hardware accelerator has been developed to run modified versions of the HiCut and HyperCut packet classification algorithms. This section starts by first explaining the HiCut and HyperCut algorithms and then explains the modifications made to them in order to make them better suited to hardware acceleration.

4.1 Hierarchical Intelligent Cuttings (HiCut)

HiCuts by Gupta and McKeown [2] is a decision-based tree algorithm, which allows incremental updates to a ruleset. It takes a geometric view of packet classification by considering each rule in a ruleset as a hypercube in hyperspace defined by the F fields of a packet header. The algorithm constructs the decision tree by recursively cutting the hyperspace one dimension at a time into sub regions. These sub regions will contain the rules whose hypercube overlap. Each cut along a dimension will increase the number of sub regions with each sub region containing fewer rules. The algorithm will keep cutting into the hyperspace until none of the sub regions exceed a predetermined number called *binth*. The number of cuts *np* which can be performed on a dimension at an internal node i is limited using a predefined variable known as *spfac* in order to prevent the decision tree becoming too fat causing a memory explosion. Each cut creates a child node, with the number of cuts always starting with 2 and doubling each time the following equation is satisfied:

$$spfac * number\ of\ rules\ at\ i \leq \sum rules\ at\ each\ child\ of\ i + np \qquad (4)$$

One method used for deciding the dimension to cut is to record the largest number of rules contained in a child after cutting each dimension and pick the dimension that returns the smallest number. Each time a packet arrives, the tree is traversed from the root node until a leaf node is found, which stores a small number of rules limited by the *binth* value. Once a leaf node is reached a small linear search of the rules contained within it is performed to find the matching rule. An example decision tree is shown in Figure 5 (A).

4.2 Multidimensional Cutting (HyperCuts)

HyperCuts by Singh et al [3] is a modification of the HiCuts algorithm, which also allows incremental updates. The main difference from HiCuts is that HyperCuts recursively cuts the hyperspace into sub regions by performing cuts on multiple dimensions at a time. The algorithm chooses dimensions for cutting at an internal node by calculating the mean number of

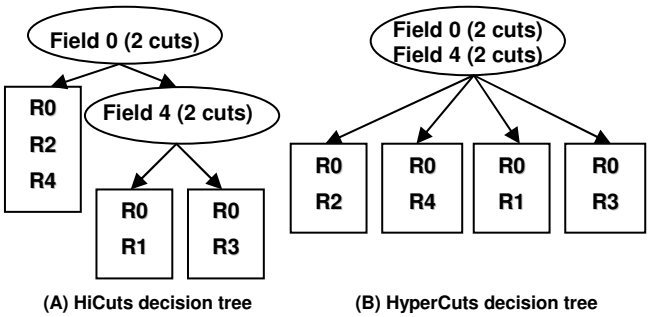

Figure 5. Example decision trees with an ellipse denoting an internal node and a rectangle denoting a leaf node

distinct range specifications for all dimensions and choosing the dimensions whose number of distinct range specifications is greater than or equal to the mean number of range specifications. The algorithm also limits the number of cuts which can be performed to an internal node *i* using a space measure function in order to prevent memory explosion. The maximum number of child nodes created by the combination of cuts between the chosen dimensions is bound by the following condition:

$$\textit{max child nodes at } i \leq \textit{spfac}*\textit{sqrt}(\textit{number of rules at } i) \quad (5)$$

The combination of cuts between the chosen dimensions which result in the smallest number of max rules stored in a child node is used. HyperCuts takes advantage of extra heuristics when building the search structure. One of these is region compaction which allows for more efficient cutting of a dimension by only cutting the region covered by the rules rather than the full region. Another is pushing common rule subsets upwards to reduce the replicated storage of rules by storing rules common to all child nodes in their parent node. HyperCuts and HiCuts reduce storage further by merging child nodes which have associated with them the same set of rules and removing child nodes which contain no rules. An example decision tree is shown in Figure 5 (B).

4.3 Algorithmic Changes

In order to make the algorithms better suited to hardware acceleration and consume less power during the building of the search structure, some modifications were made. The first modification was to remove the region compaction and push common rule subsets upwards heuristics from the HyperCuts algorithm. The region compaction heuristic was removed as it needed large amounts of hardware resources in order to carry out the floating point division required when calculating which path to follow when traversing the decision tree. Floating point division would also consume extra power. Pushing common rule subsets upwards was removed as it meant the searching of rules would have to be carried out while traversing the decision tree, slowing down the hardware accelerator.

The number of cuts allowed to internal nodes for both the HighCut and HyperCut algorithms is limited to 32, 64, 128 or 256 cuts. It was found through the testing of different size and shaped rulesets generated using ClassBench that 32 cuts is a much better starting position than 2, as it leads to a significant decrease in computation and causes an insignificant increase to memory consumption. It was also found that by capping the number of cuts to 256, savings are made in memory consumption and computation, with little decrease in throughput. Reducing the amount of computation will lead to power savings as less time is spent building the search structure. For HiCuts the number of cuts to an internal node starts at 32 and doubles each time the following condition is met:

$$(\textit{spfac}*\textit{number of rules at } i \leq \sum \textit{rules at each child of } i + np) \;\&\; (np<129) \quad (6)$$

HyperCuts considers dimensions for cutting with a number of distinct range specifications greater than or equal to the mean number of distinct range specifications for all the five dimensions. All combination of cuts between the chosen dimensions are considered if they obey the following condition where *spfac* can be 1, 2, 3 or 4:

$$(np \leq 2^{(4+spfac)}) \;\&\; (np \geq 32) \quad (7)$$

Capping the number of cuts to 256 also makes the algorithms better suited to hardware acceleration as all the information needed for an internal node can fit fully in one memory word, which can be accessed in a single clock cycle. Each of the cuts to an internal node requires 10 bits for the address location of the node in the search structure and 6 further bits for indicating the node type. A number between 0 and 47 for these 6 bits means that the node is a leaf node and gives the starting position for this node on the memory word at its address location, while a number other than this indicates the node is an internal node.

In order to calculate which cut the packet should traverse to, the internal node stores 8-bit mask and shift values for each dimension. The masks indicate how many cuts are to be made to each dimension, while the shift values indicate each dimension's weight. The cut to be chosen is calculated by ANDing the mask values with the corresponding 8 most significant bits from each of the packet's 5 dimensions. The resulting values for each dimension are shifted by the shift values with the results added together giving the cut to be selected.

Another modification made is to store the actual rule in the leaf node rather than a pointer. This was found during testing of the many rulesets created using ClassBench to have only a small increase in memory consumption for a large increase in throughput as data is presented to the hardware accelerator one clock cycle earlier. Each saved rule uses 160 bits of memory. The Destination and Source Ports use 32 bits each with 16 bits used for the min and max range values. The Source and Destination IP addresses use 35 bits each with 32 bits used to store the address and 3 bits for the mask. The storage requirement for the mask has been reduced from 6 to 3 bits by encoding the mask and storing 3 bits of the encoded mask value in the 3 least significant bits of the IP address when the mask is 0-27. The protocol number uses 9 bits with 8 bits used to store the number and 1 bit for the mask. The rule number uses 17 bits. Each 7704-bit memory word can hold up to 48 rules, and it is possible to perform a parallel search of these rules in one clock cycle.

In order to reduce memory consumption the nodes are rearranged after the search structure has been built. All the internal nodes are stored first followed by the leaf nodes. This modification means that the leaf nodes can be saved contiguously in the search structure, improving the storage efficiency of rules. To locate a leaf node the number of the memory word where it is located and the starting position of the leaf node within that memory word is

Table 3. Search structures generated using the modified HyperCuts algorithm for ACL1, FW1 and IPC1 rulesets generated using ClassBench (memory=bytes)

Rules	spfac	binth	speed	cycles	memory
Search Structures Built For Cyclone 3					
ACL5000	4	96	1	3	130,005
ACL10000	4	96	1	3	286,011
ACL15000	4	96	1	3	440,091
ACL20000	3	192	1	5	486,315
Search Structures Built For ASIC and Stratix 3					
ACL5000	4	96	1	3	130,005
ACL10000	4	96	1	3	286,011
ACL15000	4	96	1	3	440,091
ACL20000	4	144	1	4	495,945
ACL24920	4	144	1	4	641,358
Search Structures Built For Cyclone 3					
FW5000	4	48	1	2	125,190
FW10000	4	192	1	5	490,167
FW15000	2	336	0	9	478,611
FW20000	1	1,008	0	23	483,426
Search Structures Built For ASIC and Stratix 3					
FW5000	4	48	1	2	125,190
FW10000	4	96	1	3	687,582
FW15000	4	192	1	5	984,186
FW20000	4	288	0	8	980,334
FW23087	3	384	1	9	874,404
Search Structures Built For Cyclone 3					
IPC5000	4	96	1	3	127,116
IPC10000	4	96	1	3	248,454
IPC15000	4	96	1	3	483,426
IPC20000	3	192	1	5	461,277
Search Structures Built For ASIC and Stratix 3					
IPC5000	4	96	1	3	127,116
IPC10000	4	96	1	3	248,454
IPC15000	4	96	1	3	483,426
IPC20000	4	144	1	4	519,057
IPC24274	4	144	1	4	612,468

needed. Both the HiCut and HyperCut algorithms use parameters known as *spfac* and *binth* to trade off throughput against memory consumption. A third parameter we use to trade throughput against memory consumption is called *speed*. When the *speed* parameter is set to 0 the leaf nodes are stored contiguously. This means that the search structure is saved in the most memory efficient way possible but will not result in the highest possible throughput as the number of clock cycles needed to classify a packet will be:

$$\lfloor cycles \rfloor = ((z + pos)/48) + 1 + x \quad where: 0 \le pos \le 47, \quad z \ge 0 \qquad (8)$$

Where the number of internal nodes traversed to reach the leaf node is represented by x. The starting position of the leaf node in a memory word is represented by *pos* and z is the position of the matching rule in the leaf node. If the *speed* parameter is set to 1 a leaf node is only stored in a memory word with a staring position greater than 0 if:

$$RulesStoredInLeaf + pos \le 48 \qquad (9)$$

This means that there may be reduced storage efficiency as the leaf nodes may no longer be stored contiguously. Reduced storage efficiency will, however, lead to an increase in throughput as the number of cycles needed to classify a packet will now be:

$$\lfloor cycles \rfloor = (z/48) + 1 + x \qquad (10)$$

Table 3 shows the memory consumption and worst-case number of clock cycles needed to classify a packet for search structures built using HyperCuts. The rulesets used for generating these search structures were ACL1, FW1 and IPC1 rulesets generated using ClassBench. The *spfac*, *binth* and *speed* parameters used for generating the search structures are also shown. The hardware accelerator has been implemented on a Cyclone 3 FPGA using 496,384 bytes of memory. This includes the 493,056 bytes for the search structure which uses 512 memory words which are 7,704 bits wide each consuming 428 memory blocks. It also includes 3,328 bytes used for the buffer, which consists of 256 memory words that are 104 bits wide each consuming 3 memory blocks. The hardware accelerator has also been implemented on a Stratix 3 FPGA and as an ASIC using 989,440 bytes of memory. For these devices the memory used for the buffer remains the same while the search structure memory has been doubled from 512 memory words to 1024.

In Table 3 the same rulesets have been used for building the search structures for all 3 devices. It can be seen that the amount of memory available can have a big effect on the worst-case number of clock cycles needed to classify a packet for the FW1 rulesets. The FW1 ruleset with 20,000 rules needs, for example, at worst 23 clock cycles to classify a packet when 493,056 bytes of memory are available for the search structure using the Cyclone 3. This figure is reduced to 8 clock cycles when the amount of memory available for the search structure is doubled using an ASIC or Stratix 3. The available memory does not have such a large affect on the ACL1 or IPC1 rulesets.

5. LOW POWER ARCHITECTURE

The hardware accelerator has been designed to traverse an internal node of the decision tree and do a parallel comparison of up to 48 rules contained in a leaf node in 1 clock cycle. This is possible due to the fact the hardware accelerator can access a 7704-bit memory word every clock cycle. By storing the decision tree root node information in a register separate from main memory, it is possible to traverse the root node for an incoming packet while searching a leaf node for the previous packet. Carrying out these tasks in parallel has the effect of reducing the worst-case number of clock cycles by 1. This means that the hardware accelerator is able to classify a packet every clock cycle if the worst-case number of clock cycles needed to classify a packet is 2.

Before any packets can be classified by the hardware accelerator, the first step is to save the preprocessed search structure to memory. The hardware accelerator's memory structure consists of 107 memory cells which are 72-bits wide each. The search structure is saved using 72-bit memory words, which are first loaded to the buffer where they are then read by the hardware accelerator. A write enable signal is used for selecting which memory cell is to be written to, while a write address is saved into the buffer with each memory word. The write address selects which line of the selected memory cell the memory word is to be written to. The clock speed for the hardware accelerator is fixed at 32 MHz when the search structure is being saved in order to save it as quickly as possible.

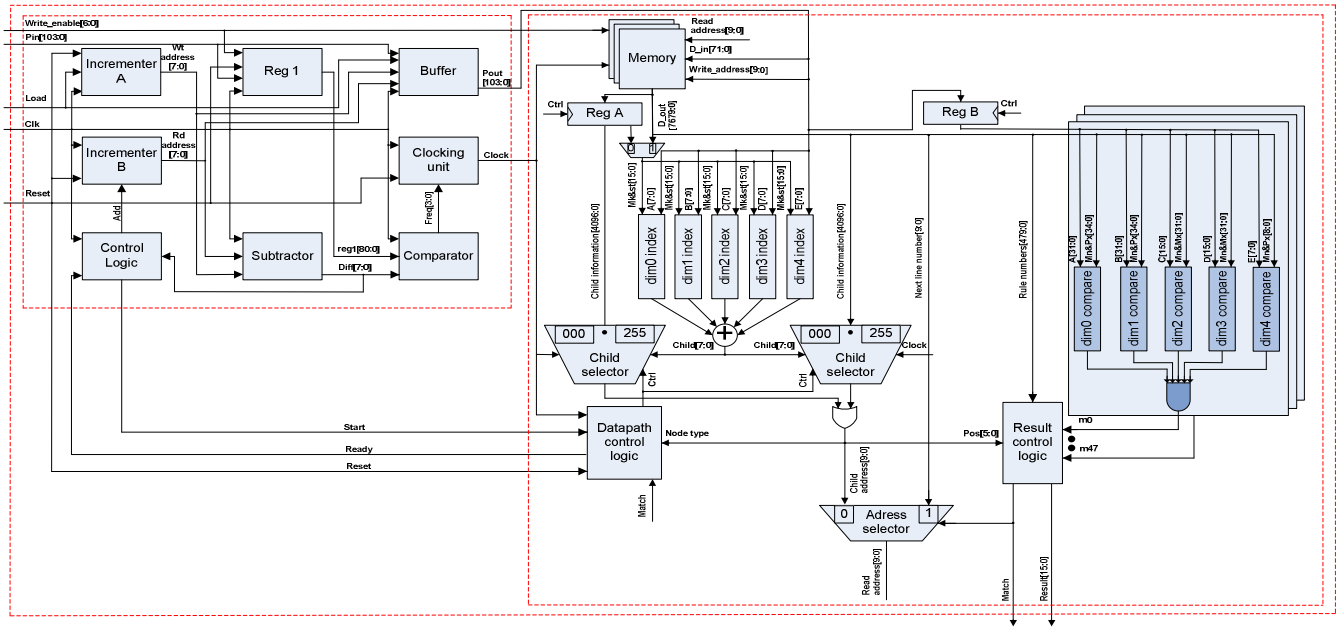

Figure 6. Low power architecture for high speed packet classification

Figure 6 shows the low power architecture for high speed packet classification. Once the Reset pin is placed low, the hardware accelerator transfers the decision tree's root node information from main memory to Reg A in 1 clock cycle. As explained in section 4.3, this information includes the starting position, memory location and node type for each of the root's child nodes. It also includes the 8-bit mask and shift values for each dimension used for selecting which child the incoming packet should go to. On the next rising clock edge the hardware accelerator begins scanning the *Start* signal from the adaptive clocking unit, which will be high if there are packets stored in the buffer. The hardware accelerator places a *Ready* signal high when this *Start* signal is high to read in a new packet from the buffer to be classified. An index value for each dimension is created by ANDing the five 8-bit mask values stored in Reg A with the 8 most significant bits from the packet's 5 dimensions read from the buffer. The resulting indexes are shifted using the 8-bit shift values stored in Reg A and then added together to determine which node address should be selected from Reg A. This node address is used to select which memory word should be loaded from main memory on the next rising clock edge. On this edge the hardware accelerator checks if the node to be loaded from main memory is an internal or leaf node.

If the node loaded from main memory is an internal node then the hardware accelerator will use the internal node information loaded to traverse to the next node. The mask values from the internal node loaded are ANDed with the packet values from the buffer. These values are shifted using the shift values from the internal node loaded and then added together. The result is used to determine which child node should be loaded from main memory on the next rising clock edge. If the selected child is still an internal node, then the process of traversing the internal nodes is repeated until a leaf node is found. Each internal node to be traversed takes 1 clock cycle.

The packet value loaded from the buffer will be transferred to Reg B if the node loaded from main memory on a rising clock edge is a leaf. The accelerator then uses 48 comparator blocks in parallel to compare the packet value in Reg B with the leaf node's rule information loaded from main memory. While this compare takes place the *Start* signal is again monitored, and if high will cause the *Ready* signal to go high, loading a new packet to be classified. The mask and index values for the root node stored in Reg A are used with the packet value loaded from the buffer, to determine which child node should be loaded from main memory once a matching rule has been found for the previous packet.

On the next rising clock edge the hardware accelerator checks if a matching rule has been found. The hardware accelerator will continue searching the leaf node if a matching rule has not been found. If a match has been found then the hardware accelerator checks to see if a packet has been loaded from the buffer. If a packet has not been loaded, then the hardware accelerator will continue monitoring the *Start* signal until it goes high. If a packet has been loaded, the hardware accelerator will check if the child node traversed to is an internal node or a leaf node. An internal node will mean repeating the process of searching for a leaf node, while a leaf node will mean repeating the process of searching for a matching rule.

6. SIMULATION RESULTS

The low power architecture for high speed packet classification was implemented in VHDL and targeted at three devices: a Cyclone EP3C120F484C8 FPGA which is built on Taiwan Semiconductor Manufacturing Company's (TSMC's) 65-nm process technology running at 1.2 Volts, a Stratix EP3SE260F1152C47 FPGA also built on TSMC's 65nm technology running at 0.9 Volts and a 65nm ASIC library by TSMC running at 1.08 Volts. The low power architecture was synthesized using Altera's Quartus 2 software for both the Cyclone 3 and Stratix 3 FPGA implementations. Post place and route timing analysis showed that timing requirements were made for the architecture implemented on both devices. The adaptive clocking unit met its timing requirement of 128 MHz and the

Table 4. FPGA Resource Utilization

Device	System A		System B	
	Logic Cells	M9K RAMS	Logic Cells	M9K RAMS
Cyclone 3	18.2%	99.8%	17.9%	99.8%
Stratix 3	5.9%	99.4%	5.8%	99.4%

hardware accelerator met its timing requirement of 32 MHz. Post place and route simulations were carried out using the Quartus 2 PowerPlay Power Analyzer Tool using VCD files generated by ModelSim. The results are explained in section 6.1.

For the ASIC solution the logic for the low power architecture was synthesized using Synopsys software. Post place and route timing analysis showed that the timing requirements for both the adaptive clocking logic and hardware accelerator logic were met. In order to estimate the power consumption for the logic the Synopsys Prime Power tool was used to analyze the annotated switching information from VCD files generated using ModelSim. Due to licensing issues the 65nm TSMC RAM compilers were not available for measuring the power consumed by memory. Instead we used the power results from RAM compilers obtained from Chartered Semiconductor manufacturing. These dual and single port RAM compilers use 130nm process technology running at 1.2 Volts. To normalize the power results for the RAM so that they were the same as the 65nm process technology running at 1.08 Volts used for the logic we used the following equation [22] where S is the scaling factor of the process technology and U is the scaling factor of the voltage:

$$P' = P * S^2 * U \qquad (11)$$

6.1 Power Results

In order to measure the power saved when using our adaptive clocking unit on our energy efficient packet classification hardware accelerator, we implemented two systems. System A used the adaptive clocking unit to run the hardware accelerator at speeds to match the traffic volume while buffering the incoming packets at a frequency of 128 MHz. It uses the same architecture described in Figure 6. System B ran the hardware accelerator at a fixed clock speed of 32 MHz while buffering the incoming packets at 128 MHz. It used the architecture shown in Figure 6 without Reg 1, the clocking unit and the comparator logic used for deciding the appropriate clock frequency. Power simulations were run for both systems implemented on the Cyclone and Stratix FPGAs and as an ASIC using the PowerPlay Power Analyzer and Prime Power tools. The resource utilization for the systems implemented on the Cyclone and Stratix 3 FPGAs can be seen in Table 4.

The simulation conditions for both systems are identical with packets read in at rates of 32, 16, 8, 4, 2, 1, 0.5, 0.25, 0.125 and 0.0625 Mpps. The search structure used needed at worst 2 clock cycles to classify a packet. This meant a packet was classified on each clock cycle when reading in 32 Mpps. The power consumption for the two systems implemented on the three technologies can be seen in Figures 7, 8 and 9. The power figures for system A are shown on the right for each packet speed and system B on the left. Looking at Figure 7 it can be seen that system A with the adaptive clocking uses 0.25% more power than system B with the fixed clock speed when implemented as an ASIC while classifying 32 Mpps. This is due to the extra logic used for the frequency scaling. It can be seen that system A shows

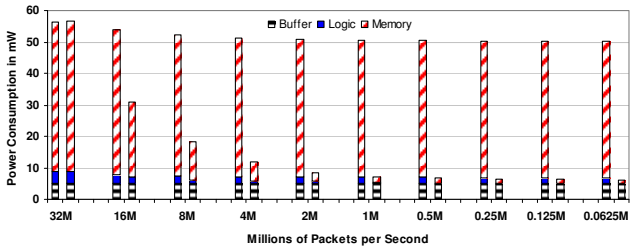

Figure 7. Power figures for ASIC implementation

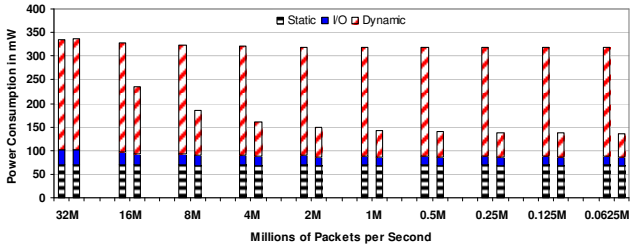

Figure 8. Power figures for Cyclone 3 implementation

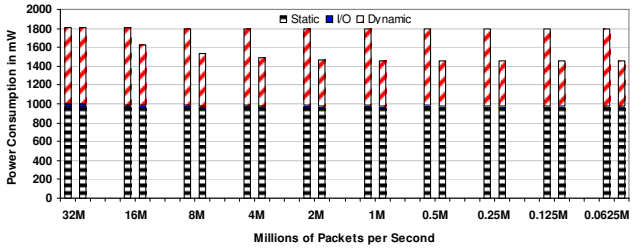

Figure 9. Power figures for Stratix 3 implementation

power savings of 89% when the packet speed drops to 0.0625 Mpps. The ASIC implementation shows good power savings, as most of the power consumed is dynamic rather than static.

Figure 8 shows power figures for the Cyclone 3 and it can be seen that system A with the adaptive clocking uses 0.7% more power than system B with the fixed clock speed when there are 32 Mpps. This is due to the fact system A uses 0.3% more of the Cyclone 3 logic resources to implement frequency scaling. System A shows power savings of 57.16% when the packet speed drops down to 0.0625 Mpps. The Cyclone 3 implementation shows lower power savings than the ASIC implementation due to the fact the FPGA has a larger percentage of its power consumption due to static power than the ASIC.

Finally Figure 9 shows the power results for the Stratix 3. When classifying 32 Mpps it can be seen that the power consumed by system A and B are almost identical, as system A only uses an extra 0.1% of the Stratix 3 logic resources to implement frequency scaling. System A shows power savings of 19% when the packet speed drops to 0.0625 Mpps. It can be seen that the power consumption is much higher for the Stratix than the Cyclone FPGA. This is because the classifier implemented on the Stratix uses double the memory of the classifier implemented on the Cyclone. The Stratix also has much more logic and memory resources available, leading to a larger amount of static power consumption. This large amount of static power is why the Stratix shows poorer reductions in power consumption.

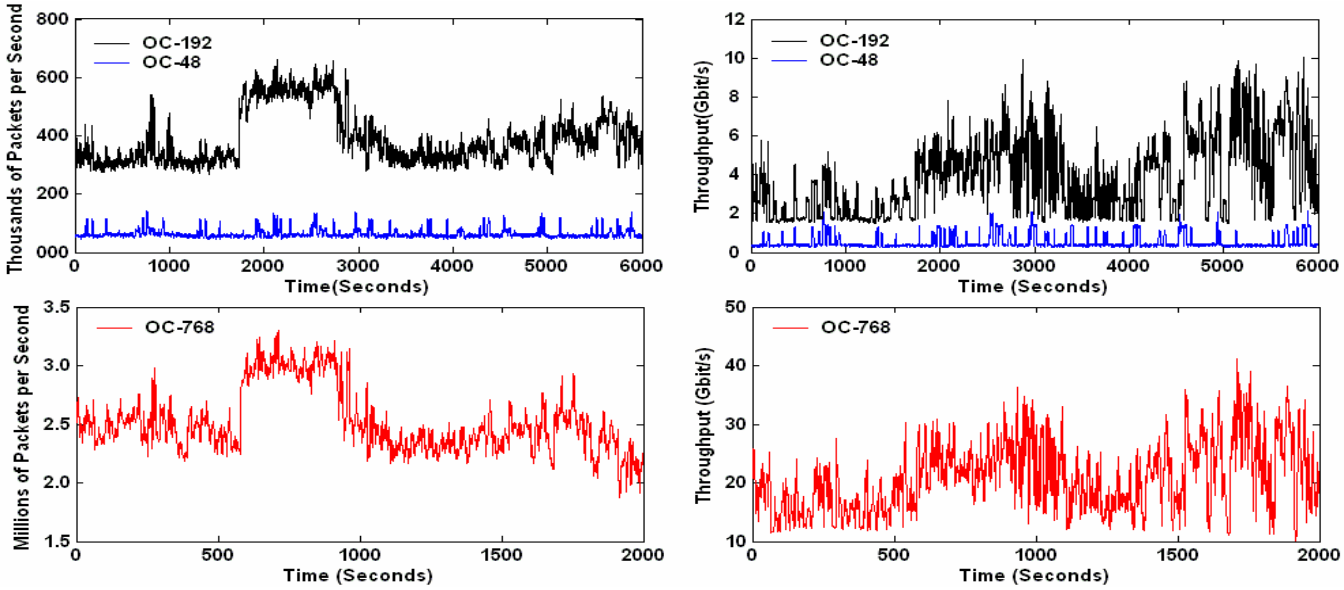

Figure 10. Throughput in Packets per Second and Gigabits per Second for the OC-48, OC-192 and OC-768 traces created

6.2 Analysis on Real Life Network Traces

In order to fully test our packet classifier, we created synthetic OC-48, OC-192 and OC-768 packet traces by aggregating Abilene, CENIC, and SCO4 backbone packet traces until peak line rates of 2.5, 10 and 40 Gb/s were reached. The traces generated are shown in Figure 10. The OC-48 and OC-192 traces were looked at over a 6000 second period. When creating the OC-768 trace the timestamps were compressed from a period of 6000 down to 2000 seconds to increase the traffic volume. The peak number of packets per second for the traces generated is 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.

To measure the power savings made by frequency scaling we used the cycle accurate simulator we developed for our low power architecture for high speed packet classification. The simulator used the power figures shown in Figures 7, 8 and 9 which were measured using the Prime Power and PowerPlay power analysis tools. The OC-48, OC-192 and OC-768 traces were run on the simulator while classifying rules using the ACL1, FW1 and IPC1 search structures built for the three devices. Details of these search structures are shown in Table 3. We spliced the time stamps from the three network traces to the packets used by the ACL1, FW1 and IPC1 rulesets generated using ClassBench.

Due to space limitations it is not possible to show all results measured. Figures 11, 12 and 13 show results for the ASIC, Cyclone 3 and Stratix 3 implementations classifying packets for the OC-48, OC-192 and OC768 traces using the search structures built for the ACL1, FW1 and IPC1 rulesets containing 20,000 rules. The results for system A with the frequency scaling described in section 6.1 are shown on the right while the results of system B using a fixed clock speed are shown on the left. The results shown in these three figures are similar to the results obtained for the other rulesets. Looking at the results for ASIC implementation, it can be seen that power savings as high as 88.8% were recorded for the OC-48 trace using the ACL1 and

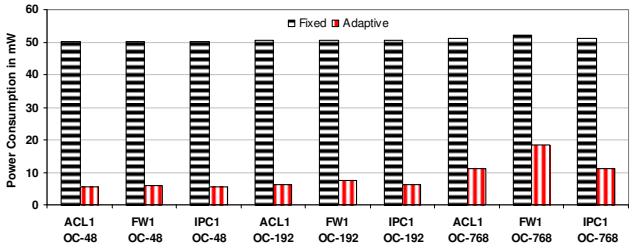

Figure 11. ASIC implementation classifying network traces using rulesets containing 20,000 rules

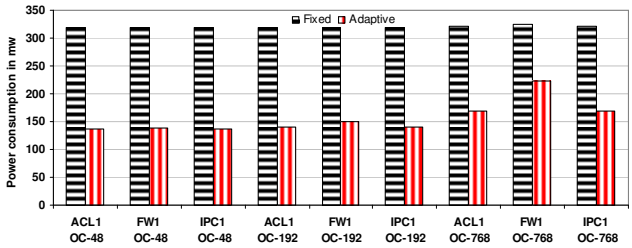

Figure 12. Cyclone 3 implementation classifying network traces using rulesets containing 20,000 rules

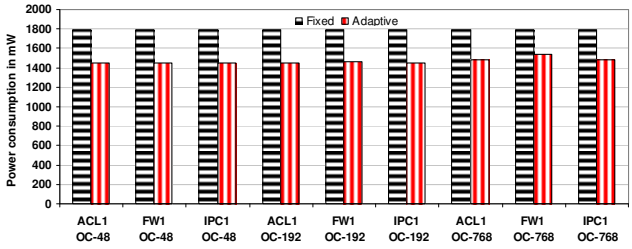

Figure 13. Stratix 3 implementation classifying network traces using rulesets containing 20,000 rules

IPC1 rulesets. The lowest power saving recorded was for the OC-768 trace using the FW1 ruleset with power savings of 64.46%. The Cyclone 3 results show a best-case power saving of 57% for the traces tested and a worst-case saving of 31.2%. For the Stratix 3 a best-case power saving of 19% was recorded and a worst-case saving of 14.2%. These results show that the adaptive clocking unit achieves large power savings on all network traces for even large rulesets when compared to a fixed clock rate.

7. Conclusion

In this paper we have presented a low power architecture for a high speed packet classifier capable of meeting OC-768 line speed for rulesets containing up to 49,000 rules. The architecture presented has been tested while classifying packets at line speeds of up to OC-768 using ACL1, FW1 and IPC1 rulesets. Simulation results show that ASIC and FPGA implementations of our low power architecture can reduce power consumption by between 17-88% by adjusting the frequency of an energy efficient hardware accelerator to match the traffic volume on a router line card.

The architecture would be ideally suited to implementation as an on-chip hardware accelerator, relieving the burden from a programmable network processor's processing engines, or as an off-chip high speed classifier on a router line card. The architecture consists of an adaptive clocking unit and a low power hardware accelerator which implements modified versions of the HiCuts and HyperCuts algorithms. The low power hardware accelerator uses SRAM rather than TCAM in order to reduce power consumption.

8. ACKNOWLEDGMENTS

This work was co-funded by the Irish Research Council for Science, Engineering and Technology, funded by the National Development Plan, the China/Ireland Science and Technology Collaboration Research Fund (2006DFA11170), NSFC (60573121, 60625201), and the Cultivation Fund of the Key Scientific and Technical Innovation Project, MoE, China (705003).

9. REFERENCES

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

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

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

[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] A. Kennedy, D. Bermingham, X. Wang, B. Liu. "Power Analysis of Packet Classification on Programmable Network Processors". *2007 IEEE Intl Conf on Signal Processing and Communications*, Dubai, 24-27 Nov, pp.1231-1234.

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

[12] 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.

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

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

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

[16] Corporation for Education Network Initiatives in California trace ftp://pma.nlanr.net/traces/long/cnic/1/

[17] Y. Luo, J. Yu, J.Yang, L. N. Bhuyan, "Conserving network processor power consumption by exploiting traffic variability", *ACM Trans. Archit. Code Optim.* 4, 1 (Mar. 2007)

[18] Ravi Kokku, Upendra B. Shevade, Nishit S. Shah, Mike Dahlin, Harrick M. Vin "Energy- Efficient Packet Processing", www.cs.utexas.edu/users/dahlin/papers/packet-power-feb2004.pdf

[19] Yan Luo, Jun Yang, Laxmi Bhuyan, Li Zhao, "NePSim: A Network Processor Simulator with Power Evaluation Framework", *IEEE Micro Special Issue on Network Processors for Future High-End Systems and Applications*, Sept/Oct 2004.

[20] C. T. Chow, L. S. M. Tsui, P. H. W. Leong, W. Luk, S. Wilton, "Dynamic voltage scaling for commercial FPGAs" , *IEEE International Conference on Field Programmable Technology*, December, 2005

[21] 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.

[22] A. Kinane, "Energy Efficient Hardware Acceleration of Multimedia Processing Tools" *PhD thesis, Dublin City University*, May 2006.