© 2015 IJSRSET | Volume 1 | Issue 1 | Print ISSN : 2395-1990 | Online ISSN : 2394-4099
Themed Section: Engineering and Technology

Merge FSM Based Low Power Packet Classification

Swapna S.R*[1] , Dr Sreeja Mole S.S[2]
Narayanaguru College of Engineering, Tamilnadu, India,

ABSTRACT

Packet classification is a vital and complicated task as the processing of packets should be done at a specified line speed. In order to classify a packet as belonging to a particular flow or set of flows, network nodes must perform a search over a set of filters using multiple fields of the packet as the search key. Packet classification is used by networking equipment to sort packets into flows by comparing their headers to a list of rules. A flow is used to decide a packet's priority and the manner in which it is processed. Packet classification is a difficult task due to the fact that all packets must be processed at wire speed and rule sets can contain tens of thousands of rules. Also the performance of today's packet classification solutions depends on the characteristics of rule sets. The range-based packet classification function maps input packets to the highest-priority matching rule in a given rule set specified by ranges. In this project, a Merge FSM model based Classifier is proposed to reduce its complexity and time consumption. The contributions of this work towards the area of packet classification are hardware accelerators that allow packet classification to be implemented at core network line speeds when classifying packets using rulesets containing tens of thousands of rules. A new pre-cutting process has been implemented to reduce the memory size to fit in an FPGA. This classifier can classify packets with high speed and with a power consumption factor of less than 3W. The proposed algorithm also removes the need for floating point division to be performed when classifying a packet, allowing higher clock speeds and thus obtaining higher throughputs.

Keywords: Packet Classification, Low Power, Accelerator, FSM, Throughput, Speed, Classification Engine

I. INTRODUCTION

Usage of internet increases day by day because of its ease of access through a wide range of devices such as desktops, notebooks, tablets, and smartphones. These results in real strain on the networking equipment needed to inspect and process the resultant traffic. A survey showed that [1] this simple access has allowed Internet penetration to reach 32.7% of the world's population by December 2011, with the number of Internet users growing by 528% between 2000 and 2011. This survey also showed that the U.S. had over 108 million internet users in 2000and in 2001, it becomes in billion ranges. Thus when considering that the total amount of energy used in the year 2000 by various networking devices in the U.S. equated to the yearly output of a typical nuclear reactor unit. This means that the current amount of energy used by networking devices worldwide could exceed the yearly output of 21 nuclear reactor units. Therefore Power consumption should be a key concern when designing any new networking equipment for solving ever increasing amount of network traffic. Network processors are key components used to process packets as they pass through a network. Main functions were packet fragmentation and reassembly, encryption, forwarding, and classification. Reducing the pressure of Network processor by addition of extra processing capacity is not easy due to factors such as silicon limitations and tight power budgets. Ramping up clock speeds to gain extra performance is difficult due to physical limitations in the silicon used to create these devices, while increasing the number of processing cores can cause difficulty when it comes to writing the software needed to control the network processors. Both these approaches also lead to large increases in power consumption due to the extra heat generated by increasing the clock speed and the extra transistors needed to increase the number of processing cores. By using of hardware accelerators dedicated to the heaviest tasks of a network processor can help to reduce power consumption while increasing processing capacity. This is because a hardware accelerator can be designed to have fewer transistors than that of the general-purpose processors used in multi-core network processors. It can also process more data than a general-purpose processor while running at slower clock speeds as they are optimized to carry out specific tasks. Large savings in power consumption can occur due to high reduction in clock speed and number of transistors.

RELATED STUDY

Large number of packet classification algorithms has been published in the past decade. Most of those algorithms fall into two categories:

Decomposition-based algorithms perform independent search on each field and eventually combine the search results from all fields. These types of algorithms are desirable for hardware implementation due to their parallel search on multiple fields. Main disadvantage is that substantial storage is usually needed to merge the independent search results in order to obtain the final result. Thus decomposition based algorithms have poor scalability, and work well only for small-scale rule sets. Decision-tree-based algorithms take the geometric view of the packet classification problem. Here each rule defines a hypercube in a *d*-dimensional space where *d* is the number of header fields considered for packet classification. Each packet defines a point in this *d*-dimensional space. The decision tree construction algorithm employs several heuristics to cut the space recursively into smaller subspaces. Each subspace ends up with fewer rules, which helps to a point a low-cost linear search to find the best matching rule.

II. METHODS AND MATERIAL

A. DECISION TREE BASED ALGORITHM

To store ten- thousands of unique rules in the on-chip memory of a single FPGA, needs to reduce the memory requirement of the decision tree. Here integrate two optimization techniques such as rule overlap reduction and precise range cuttings into the decision tree construction algorithm. Starting from the root node with the full rule set, recursively cut the tree nodes until the number of rule in all the leaf nodes is smaller than a parameter named list size. At each node, we need to figure out the set of fields to cut and the number of cuts performed on each field. Therefore restrict the maximum number of cuts at each node to be 64. In other words, an internal node can have 2, 4, 8, 16, 32 or 64 children. For the port fields, instead of the number of cuts need to determine the precise cut points. We restrict the number of cuts on port fields to be at most 2 since more bits are needed to store the cut points than to store the number of cuts,. For example, we can have 2 cuts on source addresses (SA), 4 cuts on destination addresses (DA), 2 cuts on source port(SP), and 2 cuts on destination port(DP). We do not cut on the protocol field since the first 4 fields are normally enough to distinguish different rules in real life [2].

Table 1: Example Rule Set. (SA/DA:8-bit; SP/DP: 4-bit; Protocol: 2-bit)

Rule	**SA**	**DA**	**SP**	**DP**	**Protocol**	**Priority**	**Action**
R1	*	*	2-9	6-11	Any	1	act0
R2	1*	0*	3-8	1-4	10	2	act0
R3	0*	0110*	9-12	10-13	11	3	act1
R4	0*	11*	11-14	4-8	Any	4	act2
R5	011*	11*	1-4	9-15	10	5	act2
R6	011*	11*	1-4	4-15	10	5	act1
R7	110*	00*	0-15	5-6	11	6	act3
R8	110*	0110*	0-15	5-6	Any	6	act0
R9	111*	0110*	0-15	7-9	11	7	act2
R10	111*	00*	0-15	4-9	Any	7	act1

Table 1 shows a simplified example, where each rule contains match conditions for 5fields: 8-bit source and destination addresses, 4-bit source and destination port numbers, and a 2-bit protocol value.**[Figure- 1]** shows the decision tree constructed for the rule set given in Table 1.

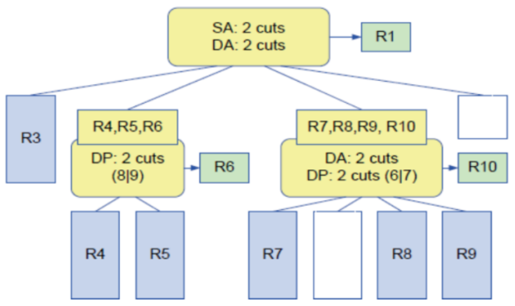


Figure 1: Building Decision Tree

Various Decision tree-based algorithms were:

1. Hicuts
2. Modular Packet Classification
3. HyperCuts

B. PACKET CLASSIFICATION USING HIERARCHICAL INTELLIGENT CUTTINGS (HICUTS)

The HiCut algorithm [2] works by carefully pre-processing the classifier to build a decision tree data structure. Each time a packet arrives, the decision tree is traversed to find a leaf node, where leaf node stores a small number of rules. By linear searching among these rules provides the desired matching. The shape and depth of the decision tree as well as the local decisions to be made at each node in the tree are chosen when the search tree is built. The following **[Figure-2]** illustrates an example of the decision-tree construction for a 2D filter set. There are five rectangles on the plane, each of them representing a filter. First step, cut is made along the *x*-axis to generate 4 sub-regions. After that, select two of these sub-regions to cut along the *y*-axis and *x*-axis,. Now each sub-region overlaps less than or equal to 2 rectangles. The cutting can be stopped, if it is affordable to do a linear search on at most 2 filters. The number of decision tree nodes and the number of stored filters determine the storage of the algorithm data structure, and the depth of the decision tree and the number of filters in the leaf nodes determine the worst-case lookup throughput.

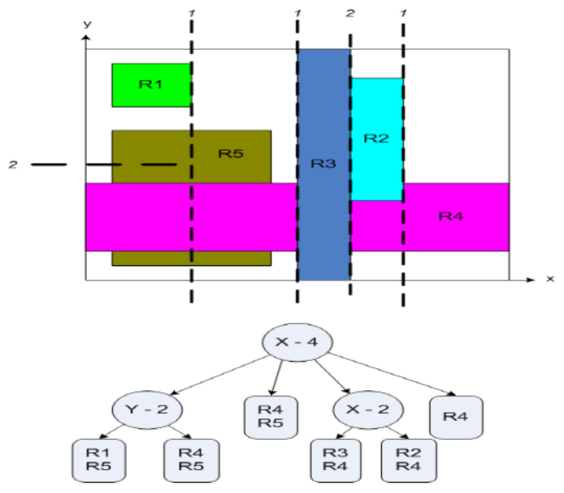


Figure 2: Decision Tree Construction For A 2D Filter Set

C. PACKET CLASSIFICATION USING MODULAR PACKET CLASSIFICATION

This algorithm [3] approaches the problem of packet classification very practically. Algorithm proposes which combines heuristic tree search with the use of filter buckets. It has high performance and economic storage requirement, algorithm is unique in the sense that it can adapt to the input packet distribution by taking into account the relative filter usage. By examining specific bit positions algorithm tries to eliminate as many filters as possible. When the set of remaining filters is less than some pre-specified maximum, instead of eliminating all terminated the first step. This set of filters is called as filter bucket. This early termination avoids steps of completely differentiate between a few “similar” filters. In the second step, the filter bucket is processed to find a match. A completely different procedure can be used due to the limited size of a filter bucket . Therefore this algorithm is a modular composition of two procedures: the first to decompose large filter table into small filter buckets of a fixed maximum size, and the second procedure is to process filter buckets of limited size to find a match.

D.PACKET CLASSIFICATION USING HYPERCUTS ALGORITHM

HyperCuts [4] is based on a decision tree structure like HiCuts. In HiCuts, each node in the decision tree represents a hyperplane. But in HyperCut each node in the decision tree represents a k--dimensional hypercube. HyperCuts can provide an order of magnitude improvement over existing classification algorithms using this extra degree of freedom and a new set of heuristics to find optimal hypercubes for a given amount of storage. HyperCuts uses less memory than HiCuts which is optimized for memory. The worst case search time of HyperCuts is 50-500% better than that of HiCuts.so HyperCuts is optimized for speed. An example of a two dimensional classifier is shown in **[Figure – 3]** with 4 rules: R1….R4. Each rule is represented by a rectangle in two dimensional space. The left figure represents the action of HiCuts. At each node HiCuts builds a decision tree using local optimization decisions to choose the next dimension of test in order to find how many cuts to make in the chosen dimension. The leaves of the HiCuts tree store a list of rules. These rules may match the search path to the leaf. The left part of **[Figure - 3]** shows how the HiCuts algorithm works on the example rule set.

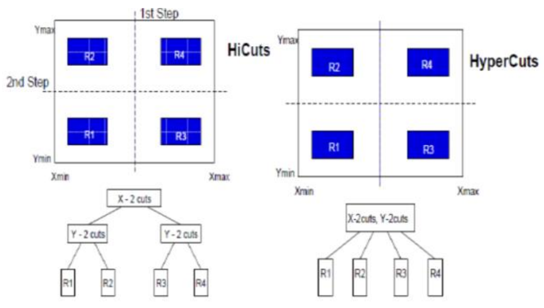


Figure 3: HiCuts Vs HyperCuts

No matter how many cuts are going to be executed at a time, assuming the maximum number of rules held in a leaf is 1. HiCuts algorithm requires at least two levels in the decision tree. By introducing one more degree of freedom HyperCuts algorithm eliminates this limitation in HiCuts. Each node in the decision tree represents a decision taken on the most representative dimensions, as opposed to using only a single dimension. For each of the chosen dimensions, the number of cuts is computed based on conditions dependent on the amount of space that is available for the search structure. In the example in **[Figure – 3]** Hyper-Cuts (on the right) cuts the plane into four squares with one direct cut, reducing the height of the decision tree to 1.

MERGE FSM

Merge FSM model based Classifier is proposed to reduce its complexity and time consumption. The contributions of this work towards the area of packet classification are hardware accelerators that allow packet classification to be implemented at core network line speeds when classifying packets using rulesets containing tens of thousands of rules. A new pre-cutting process has been implemented to reduce the memory size to fit in an FPGA. This classifier can classify packets with high speed and with a power consumption factor of less than 3W. The proposed algorithm also removes the need for floating point division to be performed when classifying a packet, allowing higher clock speeds and thus obtaining higher throughputs. Computer-aided synthesis of sequential circuits is an active area of research as many systems treat the control unit of the designed circuit as a finite state machine realized with some regular structure.

As its name indicates a Finite state machine (FSM) is a circuit with internal states. Finite-state machines provide a simple computational model with many applications. Recall the definition of a Turing machine: a finite-state controller with a movable read/write head on an unbounded storage tape. If we restrict the head to move in only one direction, we have the general case of a finite-state machine. The sequence of symbols being read can be thought to constitute the input, while the sequence of symbols being written could be thought to constitute the output. Finite-state machines, also called finite-state automata (singular: automaton) or just finite automata are much more restrictive in their capabilities than Turing machines. For example, we can show that it is not possible for a finite-state machine to determine whether the input consists of a prime number of symbols. Much simpler languages, such as the sequences of well-balanced parenthesis strings, also cannot be recognized by finite-state machines. Still there are the following applications:

- Simple forms of pattern matching (precisely the patterns definable by "regular expressions", as we shall see).
- Models for sequential logic circuits, of the kind on which every present-day computer and many device controllers are based.
- An intimate relationship with directed graphs having arcs labeled with symbols from the input alphabet.

Even though each of these models can be depicted in a different setting, they have a common mathematical basis.

Finite-state machines can model a large number of problems, among which are electronic design automation, communication protocol design, language parsing and other engineering applications. In biology and artificial intelligence research, state machines or hierarchies of state machines have been used to describe neurological systems and in linguistics—to describe the grammars of natural languages.

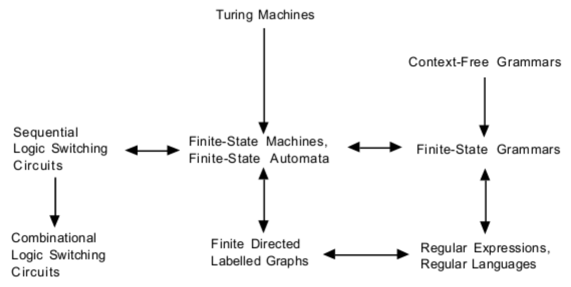

Turing Machines
Context-Free Grammars
Sequential Logic Switching Circuits
Finite-State Machines, Finite-State Automata
Finite-State Grammars
Combinational Logic Switching Circuits
Finite Directed Labelled Graphs
Regular Expressions, Regular Languages

Figure 4: Finite State Machine as Turing Machine

A. Finite-State Machines as Restricted Turing Machines:

One way to view the finite-state machine model as a more restrictive Turing machine is to separate the input and output halves of the tapes, as shown below. However, mathematically we don't need to rely on the tape metaphor; just viewing the input and output as sequences of events occurring in time would be adequate.

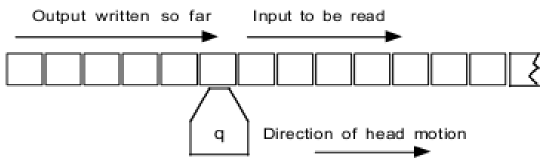


Figure 5: Finite-State Machines as Restricted Turing Machines

B. Modeling the Behavior of Finite-State Machines:

Concentrating initially on transducers, there are several different notations we can use to capture the behavior of finite-state machines:

- As a functional program mapping one list into another.
- As a restricted imperative program, reading input a single character at a time and producing output a single character at a time.
- As a feedback system.
- Representation of functions as a table
- Representation of functions by a directed labeled graph

For concreteness, we shall use the sequence-to-sequence model of the machine, although the other models can be represented similarly.

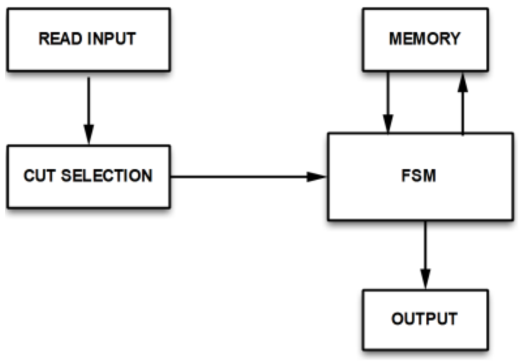


Figure 6: Block Diagram For Merge FSM Based Packet Classification

PACKET CLASSIFICATION ENGINE

Figure below shows the architecture of the packet classification engine which is built using two modules. The first module 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 module 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 with no rule matched. The leaf node searcher employs 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.

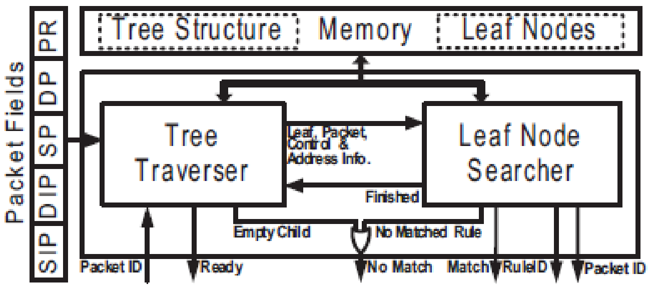


Figure 7: Architecture used by the packet classification engines

A. ARCHITECTURE OF THE CLASSIFIER

The classifier has been implemented with multiple packet classification engines working in parallel using Stratix III and Cyclone III FPGAs. The maximum clock speed that an engine can achieve when implemented using an FPGA is much slower than the maximum clock speed of an FPGAs internal memory. This is due to logic delays in the components used by an engine such as the comparator blocks. It is, therefore, necessary to use multiple engines working in parallel so that the classifier can achieve maximum throughput. The use of multiple engines will help to ensure that the bandwidth of an FPGAs internal memory is better utilized. Another reason for using multiple packet classification engines working in parallel is that it allows rulesets that contain many wildcard rules to be broken up into groups, with each engine used to search a group for a matching rule. Splitting rulesets that contain many wildcard rules into groups makes it easier to build shallow decision trees that have small leaf nodes, which helps to increase throughput and reduce memory usage. This is because the rules with wildcard source IP addresses can be kept in one group and the rules with wildcard destination IP addresses can be kept in another group. The group that contains the wildcard destination IP addresses is cut by performing the majority of the cuts to the source IP address, while the group that contains the wildcard source IP addresses is cut by performing the majority of the cuts to the destination IP address. The majority of the cuts are usually performed to the IP addresses because many of the ports can contain the common port range of 1024 to 65 535. The matching rule with the highest priority (rule with the lowest rule ID) will be chosen in the case where multiple engines return a matching rule. The search structure for each group can be saved to the same block of memory that is shared by the engines. Both the Stratix III and Cyclone III implementations of the classifier use eight packet classification engines working in parallel.

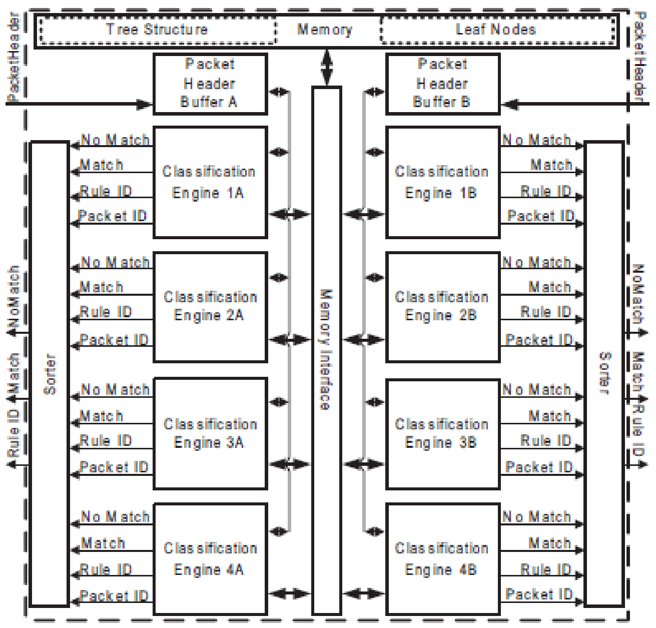


Figure 8: Architecture of the classifier.

Figure above shows the classifier's architecture, which takes advantage of the fact that the internal memory of an FPGA is dual port by placing two separate classifiers in parallel, sharing the same memory. Each classifier reads data from a separate data port and has its own packet buffer for storing the headers of incoming packets, four engines that work in parallel to maximize the bandwidth usage of a data port and a sorter logic block used to make sure that the classification results are outputted in the correct order. The packet buffer stores the source and destination IP addresses, source and destination port numbers, and protocol number from 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 Stratix III implementation of this classifier has 46 080 memory words available to save the search structures required for classifying packets, while the Cyclone III implementation has 12 288 memory words available. 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 ruleset 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 rulesets containing a large number of wildcard rules. The sorter logic block is used to make sure that the matching rule 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 rulesets are broken up into groups. The sorter logic block accepts the Match, NoMatch, RuleI D, 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 NoMatch 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, NoMatch, and RuleI D signals for a classified packet to a chain of registers andmultiplexers in series. The register selected will depend on the packet ID number. The Match, NoMatch, and RuleI D 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 classifierin the same order that the packets were inputted.

III. RESULTS AND DISCUSSION

The classifier has been tested extensively by measuring its logic and memory usage, throughput in terms of Mpps, amount of memory it requires when storing the search structures needed to classify packets for access control list (ACL), firewall (FW), and Internet protocol chain (IPC) rulesets generated using ClassBench , worst case number of memory accesses needed to classify a packet, power consumption, and its performance when classifying packets using real life OC-48, OC-192, and OC-768 packet traces. These results have been benchmarked against state-of-the-art dedicated FPGA based classifiers.

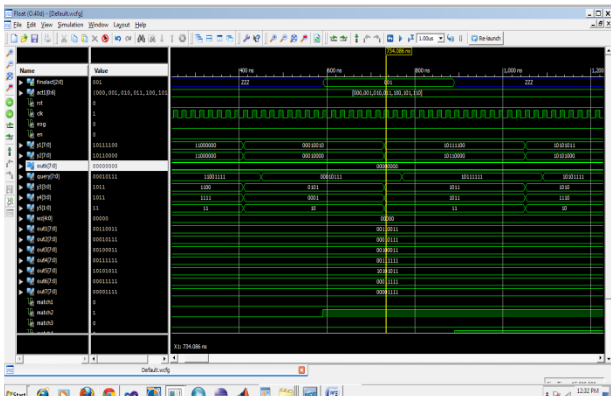
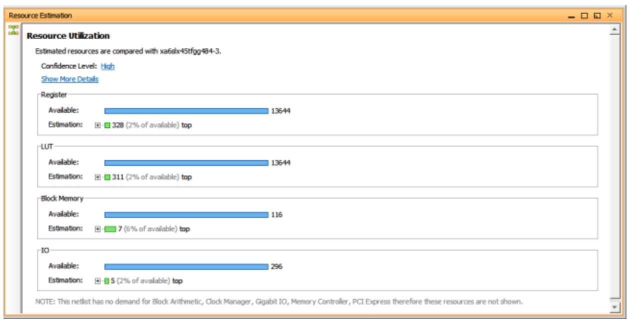


Figure 9: Matched Output


Resource Estimation

Resource Utilization

Estimated resources are compared with xa6slx45tfgg484-3.

Confidence Level: High

Show More Details

Register

Available: 13644

Estimation: 328 (2% of available) top

LUT

Available: 13644

Estimation: 311 (2% of available) top

Block Memory

Available: 116

Estimation: 7 (6% of available) top

IO

Available: 296

Estimation: 5 (2% of available) top

NOTE: This netlist has no demand for Block Arithmetic, Clock Manager, Gigabit IO, Memory Controller, PCI Express therefore these resources are not shown.

Figure 10: Memory Usage

IV. CONCLUSION

The results show that the classifier performs well in terms of memory usage and worst case number of memory accesses. Research into the increased throughput of packet classification through hardware acceleration with power consumption in mind is an increasingly important field of research as hardware accelerators have become essential when trying to meet core network line speeds. This is because line speeds are growing steadily due to advances in optical fiber technology and rulesets are expanding due to the increasing number of services that need to be performed.

V. REFERENCE

[1]. Usage and population statistics (2012,jun.) [online].Available:http://www.internetworldstats.com/stats.htm

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

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

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

[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]. 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.

[7]. V. Srinivasan, S. Suri, and G. Varghese, “Packet classification using tuple space search, ” in Proc. ACM Special Interest Group Data Commun.Conf., Sep. 1999, pp. 135–146.