A Hardware-Engine for Layer-2 classification in low-storage, ultra-high bandwidth environments

V. Papaefstathiou, I. Papaefstathiou
Foundation of Research & Technology Hellas (FORTH),
Institute of Computer Science (ICS),
Vassilika Vouton, GR71110, Heraklio, Crete, Greece
{papaef, ygp}@ics.forth.gr

Abstract: *- Ethernet is the most common Layer-2 network protocol, and it is currently being deployed beyond the tight borders of LANs. In order to accommodate the needs of MANs and WANs, several QoS mechanisms employed at the MAC sublayer of Ethernet have been proposed. These QoS mechanisms require identification of network flows and the classification of Ethernet packets according to certain Ethernet header fields. In this paper, we propose a classification engine employed at the MAC sublayer which uses an innovative hashing scheme and internal replacement of MAC Vendor IDs; the Hash Based Classification Engine (HBCE) compacts the tables containing the rules associated with certain MAC addresses and supports extremely high speed decisions –at a rate of more than 100Gb/sec-, while its memory needs are significantly lower compared to those of the similar schemes currently used. This engine has been implemented in hardware utilizing less than 0.1mm² in a state of the art CMOS technology. As a result HBCE is a very promising candidate for the next-generation Ethernet equipments that need to support classification at Data Link Layer at multi-Gigabit per second network speeds, whereas due to its very low memory requirements and low implementation complexity, it can also be employed very efficiently in lower-bandwidth wireless environments that utilize MAC mechanisms.*

1 Introduction

Ethernet is, by far, the most common Layer-2 network protocol. Mainly due to its very good cost-performance ratio, it is currently making a breakthrough in MAN and WAN networks. The deployment of MAC-based Multi-Gigabit wired or Lower-bandwidth wireless networks, and their use beyond the tight borders of LANs, motivated the development of QoS mechanisms in the MAC layer; such examples are the VLAN scheme [1], or certain QoS protocols for wireless environments [2]. Those mechanisms require identification of network flows and classification of Ethernet packets according to their MAC addresses, VLAN IDs or port numbers. Moreover, in order to be able to support fine-grained QoS they incorporate tens of thousands of independent network flows identified by the MAC, VLAN and/or Port fields. The length of the MAC addresses, namely 48-bits, is what makes the classification task difficult since exact matches in such a wide value is not a trivial task. Since the main advantage of the Ethernet networks, and the associated equipments, is their low cost, the classification solutions that would be used within the specified frameworks should be as cost efficient as possible.

The three header fields used in Ethernet switching are the MAC address, the port of the switch and the Virtual LAN (VLAN) tag; these elements are also used by the mechanisms that provide QoS in Layer-2.

MAC Address: The MAC address is a 48-bit (6 bytes) value that uniquely identifies a Network Interface Card (NIC). The first 24-bits (3 bytes) of the address identify the vendor of the card and the last 24-bits identify the card itself.

Port: A number that uniquely identifies the physical interface of the equipment (for example a 10-bit number for a 1024-port system).

VLAN: VLAN tagging was introduced in IEEE 802.1q [1] and defines how an Ethernet frame is tagged with a VLAN ID. VLAN tagging provides a mechanism to handle time-critical network traffic by setting transmission priorities to outgoing frames according to IEEE 802.1p [3]. Moreover VLANs allow network stations to be assigned to logical groups and communicate across multiple LANs as if they were on a single LAN; Ethernet bridges and switches should forward the VLAN-tagged frames only to ports that serve the specified VLAN.

In this paper, we propose a classification engine utilized at the MAC sub-layer which uses a new hashing scheme and internal replacement of MAC Vendor IDs; the Hash Based Classification Engine (HBCE) can reach classification decisions at extremely high speeds while its main advantage comes from the fact that it utilizes less than two thirds of the memory needed by the existing solutions. The efficiency of the proposed engine comes from the fact that the hashing and the replacement schemes used, take advantage of the individual characteristics of the MAC addresses. This engine has been implemented in hardware and while its implementation cost is minimal, it supports network rates higher than 100 Gb/sec while incorporating 64K independent rules.

2 Related Work

L2 classification requires the fields mentioned in the last section to be examined and the appropriate action to be performed. Therefore, the network equipments need to store

3-9810801-0-6/DATE06 © 2006 EDAA

some information and consult them for their decisions. The information regarding the MAC addresses, the VLANs and the Ports are stored in internal data structures and for each packet a search is conducted using the packet header fields.

Switches and bridges have very often integrated hardware modules handling such classification tasks; they place the various tables, holding the data structures, in internal or external memories and all operations access those tables in order to examine whether an exact match exists or not. Today's switches support up to 32K-64K MAC-address rules [4] and 4096 VLANs, hence the size of memories is relatively small.

The nature of L2 classification requires exact matches and many implementations use CAMs that provide single access matching [5]. CAM solutions, although simple, they are expensive and consume large amounts of power. Trie based solutions [6] have poor performance since they cannot handle efficiently long matching strings such as the MAC address. Moreover, trie based solutions may require several memory accesses and massive storage for the associated pointers.

Another popular solution is hashing of the MAC address bits [7] and storing the data in SRAM based lookup tables. The 48-bits are hashed using a specific hashing function and an index for the lookup table is generated. Possible collisions due to hashing are usually resolved with linked lists of entries. Hashing 48-bits into a small, say 16-bit, value requires a good function that generates differentiated values usually by taking into account all the information bits. Many solutions use the CRC polynomials for hashing since they have been proved very efficient [8] however others, mainly due to cost reasons, use direct mapping by the least significant bits of the MAC address [9].

3 Hash Based Classification Engine

Our solution for L2 classification is based on hashing, like the majority of similar products, but we propose a scheme that exactly matches the special characteristics of the MAC addresses. Moreover, our Hash Based Classification Engine (HBCE) employs internal MAC Vendor replacement. HBCE is designed to support tens-of-thousands of MAC-address rules and a couple of thousands of VLANs and port-based rules. Every rule in HBCE is associated to a number called FlowID (which can, for example, be a pointer to another memory which holds the associated information for this rule or simply a number identifying the output port of the device). We decided to use 15-bit FlowIDs, translating to 32K unique and independent network flows, which have been proved to be enough for most Ethernet equipments.

The most important part of our scheme is the lookup scheme for the MAC-address rules. The length of the MAC address, is what makes this part the most critical in terms of both speed and storage. VLANs and ports are relatively small and can be directly mapped into tables, as it is described in the next sections.

3.1 MAC Address Hashing

We developed a hashing function to map the MAC-address rules into a table that will hold the FlowID of the associated rule. Those rules are stored in a 64K table called MAC_TBL and the indexes to it are generated by our hashing function applied to the MAC address bits. The collisions due to hashing are handled by pointers to variable size blocks. Handling variable size blocks requires a dynamic memory management scheme which is described in the next sections. The number of entries in each variable size block is defined by the number of rules that collide within a specific entry of the MAC_TBL.

Our hashing scheme applies an XOR function to all 48-bits of the MAC address and the 16-bit MAC_TBL address is produced as follows:

$$MAC\_TBL_{index} = \{MAC[47:40] \textbf{ xor } MAC[31:24] \textbf{ xor } MAC[15:8], MAC[39:32] \textbf{ xor } MAC[23:16] \textbf{ xor } MAC[7:0]\}$$

To identify a certain MAC-address rule within a particular table entry we also need to save some additional information so as to be able to distinguish those that collide. Fortunately, we don't need to save all 48-bits and we take advantage of the fact that the XOR function can be "inversed". Therefore a certain MAC-value associated with address A of MAC_TBL can be reproduced by the 16-bits of A and the last 32-bits (H_{val}) of the MAC address as follows:

$$MAC[47:40] = A[15:8] \textbf{ xor } H_{val}[31:24] \textbf{ xor } H_{val}[15:8]$$
$$MAC[39:32] = A[7:0] \textbf{ xor } H_{val}[23:16] \textbf{ xor } H_{val}[7:0]$$
$$MAC[31:0] = H_{val}(31:0)$$

So by using H_{val} we can uniquely identify each MAC-address rule. If we use CRC-16 to produce the 16-bit indexes, like the popular schemes described in the related work section, then we would have to store the complete 48-bits of the MAC address since there is no inverse CRC function. Moreover, CRC polynomials don't have one-to-one correspondence between input and generated values. The speed and storage performance of our hashing function is discussed in section 4.

3.2 MAC Vendor Replacement

The official IEEE OUI [11] has published all the assigned 24-bit MAC vendor IDs and the associated company names. Based on them we have observed that the 24-bit vendor address space of the MAC addresses is not fully occupied. In fact, fewer than 8000 vendors are active instead of the 2^{24} possible. Therefore we can replace the 24-bit vendor ID with a 13-bit internally assigned vendor ID. The last 24-bits of the MAC address that uniquely identify a device, of a certain vendor, remain unchanged. This replacement reduces the storage requirements for each MAC-address rule, at the cost obviously of the replacement operation. Consequently, every incoming MAC-address rule need to be translated before the actual processing begins.

We can now consider that each MAC-address rule handled by our system is 37-bits long. Naturally, this replacement means that we keep a small table with 8192 entries called VID_RPL that matches the existing 24-bit Vendor ID values with the internally assigned 13-bit

Vendor ID values. This table can be easily constructed since all Vendor IDs are sequentially assigned by IEEE and a few 'holes' that exist in the address space can be handled by a 24-to-13 decoder. Although this table is constant and thus can be kept in a ROM, we can also use a method that learns the connected MAC addresses and assigns incrementally an internal Vendor ID.

After this replacement we define a new hashing function on the 37-bits of the MAC address. Now, the 16-bit indexes in MAC_TBL are generated as follows:

$$MAC\_TBL_{index} = \{\ MAC[31:24]\ \textbf{xor}\ MAC[15:8]\ ,\ MAC[23:16]\ \textbf{xor}\ MAC[7:0]\ \}$$

Notice that we don't use the 6 MSB of the replaced Vendor ID in order to have a byte balanced hashing function. The new H_{val} is now 21-bits and is defined as follows:

$$H_{val} = \{\ MAC[36:24]\ ,\ MAC[7:0]\ \}$$

Now, a MAC-address associated with address A of MAC_TBL can be reproduced by the 16-bits of the address and H_{val} as follows:

$$\begin{aligned} MAC[36:24] &= H_{val}[20:8] \\ MAC[23:16] &= A[15:8]\ \textbf{xor}\ H_{val}[7:0] \\ MAC[15:8] &= A[15:8]\ \textbf{xor}\ H_{val}[15:8] \\ MAC[7:0] &= H_{val}(7:0) \end{aligned}$$

3.3 MAC_TBL and Data Structures

The MAC_TBL has 64K entries. The indexes to the MAC_TBL are generated by our hashing function and therefore collisions may occur. In order to resolve these collisions efficiently, we define a complex data structure associated with each entry of the MAC_TBL. In general, we have to fully identify a MAC address associated with a certain entry of the table (by using the H_{val} field as described in the last section) while we would also like to be able to retrieve the corresponding 15-bit Flow ID. In the case where only one MAC-address rule is saved in a table entry we can save the FlowID in the 15 MSB of the word and H_{val} in the 21 LSB. If we use on-chip memories the word size is probably not a problem but in case of off-chip memories this is a critical aspect. Fortunately, the majority of the existing SRAM modules support 36-bits words and thus the Hval and the FlowID can both be fitted in exactly one memory word and without any memory overhead (i.e. empty space).

In another case, a table entry might be empty which means that it is not mapped to any MAC-address rule; we reserve the Flow ID number 0 for this purpose. Moreover, a table entry may be mapped to many MAC-address rules. In this case, where collisions occur, we have to store a pointer to the variable size block and the number of rules that collide. The number of colliding rules also indicates the size of the block. For the collisions' case we have reserved the Flow ID number 1. When a collision occurs, the least significant 17-bits of the word hold the pointer to the variable size external block and the remaining 4-bits are used to keep the number of MAC-address rules mapped to this particular table entry. 4-bits are enough for the maximum number of collisions of our system as explained below. The format of the memory words in each case is shown in Figure 3-1.

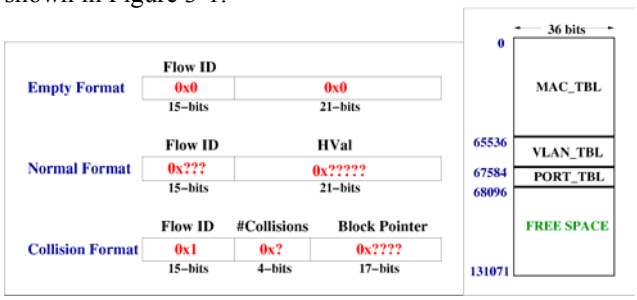

Figure 3-1 MAC_TBL entries format and memory organization

The variable size blocks also use 36-bit memory words while the different formats of their entries are identical to those depicted in Figure 3-1. An example that shows the complete data structure for an indicative set of MAC-address rules is depicted in **Error! Reference source not found.**.

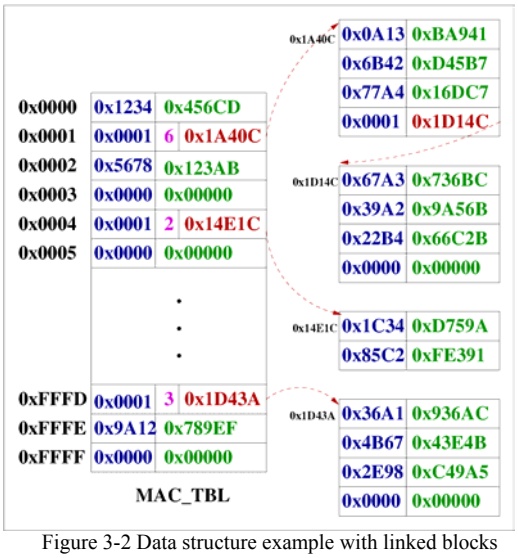

Figure 3-2 Data structure example with linked blocks

3.4 VLAN and Port Tables

Handling the VLAN tag and the Port field is simple and requires the storage of the 15-bit Flow ID associated with a certain value of each of those fields. The VLAN tag is defined as a 12-bit identifier and it is mapped in a 2K-entry directly mapped table called VLAN_TBL which hosts 2 FlowIDs per word. Similarly, the port field is a 10-bit identifier and is mapped in a 512 entry table called PORT_TBL, holding again 2 FlowIDs per entry.

4 Hardware Implementation

We designed and implemented a pipelined hardware realization of our scheme in order to support very high speed classification decisions and integrate it in a real network environment. Figure 4-1 illustrates the internal organization of HBCE. The operations of the system are handled by a Main Control Block (HBCE_MCB) which receives commands from an external processor interface (PINF). Upon a reception of a command HBCE_MCB instructs the MAC_VID block to execute the vendor ID replacement. Then, in case the requested operation is based

on the MAC address, the modified MAC address is forwarded to MAC_HSH which is responsible for the hashing. When the hashing results are ready the dedicated blocks perform the appropriate actions so as to insert (HBCE_INS), lookup (HBCE_LUP) or delete (HBCE_DEL) a certain rule in/from the data structure. For VLAN or Port-based rules, the corresponding fields are sent directly from HBCE_MCB to those blocks. All those operation-blocks communicate with the external memory through the memory handler (MEM_HDLR) and the memory controller (MEM_CTRL). The MEM_HDLR implements the dynamic memory management scheme described in section 4.4 by employing several free-lists while the MEM_CTRL is the actual low level memory interface. When the final FlowID is identified, it is returned through the memory interface to the initiator of the command. Since the lookup operation is certainly the most critical and most frequent one, the whole design has been organized so as to allow for a new lookup command to be serviced at every clock cycle (in the average case).

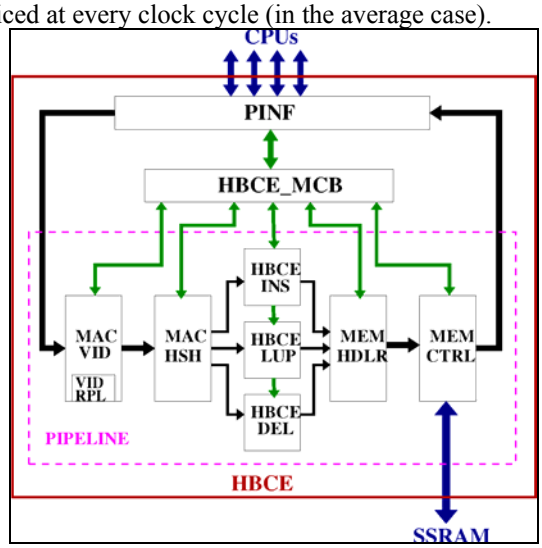

Figure 4-1 HBCE Block Diagram

4.1 Memory Organization and Tables

The current HBCE implementation is based on sequential accesses to both the external MAC_TBL and to the dynamically allocated collision nodes. Moreover, in the same external memory we have stored the VLAN table (VLAN_TBL), the Port table (PORT_TBL) and the free-lists used by the Dynamic Memory Management scheme. The memory used is 36-bits and we have used at most 128K words which have been found sufficient based on the simulations presented in [10]. The organization of this external memory is shown in Figure 3-1. The 62976 "free" memory words are used by the memory handler (MEM_HDLR) to provide dynamic allocation and deallocation of memory blocks.

4.2 Handling Dynamic Memory

The most interesting block is the MEM_HDLR which implements our dynamic memory management scheme and supports variable size blocks. It supports requests for allocation and deallocation of variable size blocks and forwards the appropriate commands to MEM_CTRL. Requests for single reads or writes to the memory are immediately forwarded to MEM_CTRL.

For our dynamic operations we use the pool of the free 61K adjacent memory words mentioned above. To support our dynamic management scheme we use a head pointer to the pool of the available memory words, a tail pointer to the last address of this pool and a current pointer to keep the state of the already used memory words. During an allocation operation we increment the current pointer. The deallocated blocks are placed into free-lists where each free-list holds all the deallocated blocks of a certain size. For every free-list we keep a head, tail pointer and a counter to keep the number of empty linked blocks. Linking between multiple blocks is implemented by writing the address of the next block in the previous block [12]. Based on the simulation results of the next section, we have decided not to support unlimited free-lists for blocks of different sizes but instead directly support blocks of 2 and 4 words that proved to provide very good results in all the scenarios examined.

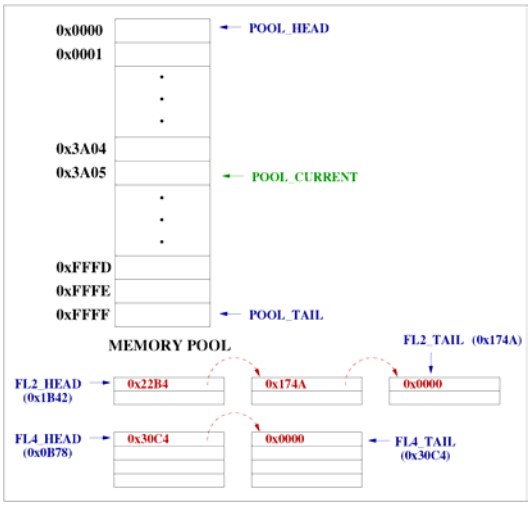

Figure 4-2 Snapshot of dynamic memory management mechanism

In case we need larger blocks, we can link internally 2 or 4-word blocks by using the "collision format" of Figure 3-1. Obviously, the main disadvantages of this implementation are that (a) if we need block sizes not multiples of 2 or 4 we have to pay a small fragmentation cost and (b) the memory overhead for the pointers needed to link a number of blocks together. Figure 3-2 depicts how multiple blocks can be linked together.

During requests for allocation of a block we first check if we have available blocks of the specified size in the corresponding free-list and if not then we take a block from the memory pool. Upon deallocation, we add the deallocated block in the tail of the corresponding free-list and increment the appropriate counter. Figure 4-2 illustrates the mechanism incorporating both the memory pool and the free-lists.

5 Performance and Hardware Cost

In this section we calculate and analyze the storage needs of HBCE and compare it with the traditional CRC-16 and direct mapped solutions, when used both synthetic and real world benchmarks. Moreover, we present the performance achieved by our hardware implementation, together with its

complexity. The indexing of MAC_TBL in our scheme is performed by the hashing function presented in subsection 3.2. As demonstrated in [10] this is indeed a very effective hash function that has similar performance to CRC-16, while it is more efficient than direct mapping; this is because the XOR function used by both CRC-16 and HBCE provides better collisions results since it generates more uniformly distributed indexes. The main advantage of HBCE, when compared with CRC, is that it requires only a small portion of the original MAC address to be stored in the corresponding table entry instead of the total 48-bits required by CRC. It is also simpler and less expensive to implement the HBCE hash function in hardware, than the corresponding CRC one.

5.1 Storage Requirements

In this subsection we demonstrate the total storage requirements of HBCE, for the large synthetic databases. The triggered collisions are handled by the dynamic memory management system described in subsection 4.4 and thus apart from the static tables used we have to calculate the number of 2-word and 4-word blocks required. The sizes of the static tables in HBCE are demonstrated in Table 5-1.

Table	Entries	Total Bytes
MAC_TBL	65536	294912
VLAN_TBL	2048	9216
PORT_TBL	512	2304
VID_RPL	8192	36864
Total	**76288**	**343296 (338 Kb)**

Table 5-1 HBCE static tables memory

In Table 5-2 we present the final storage requirements of HBCE for each database including all the collision blocks. We also present the storage requirements if CRC-16 was the hashing function for the same databases. Note that in the cases of CRC without replacement and in that of CRC with replacement (CRC-RPL) we need two 36-bit memory words for each rule because we need to keep the 48 or 37-bit internal MAC address and the corresponding 15-bit FlowID. Note also, that in the case of CRC without vendor replacement we do not need the VID_RPL table.

We can see that half megabyte is enough for HBCE to store 64K MAC-address rules. Moreover, we need 32% - 41% less memory than the equivalent CRC-16 approach. Note also that in our hardware implementation we have initially assigned 61K adjacent memory words for the variable size blocks handled by our dynamic memory management system. However, as those results demonstrate, only 70% of this space is actually used. This means that it is possible for our hardware implementation to support more than 64K MAC-address rules utilizing a total of 128K memory words; increasing the number of rules supported will, also, increase the average number of collisions.

DB Size (Active Vendors)	CRC - RPL Total (Kbytes)	CRC Total (Kbytes)	HBCE Total (Kbytes)
32K (256)	626	590	396
32K (1500)	626	590	395
32K (4000)	**626**	**590**	**396**
48K (256)	780	744	458
48K (1500)	779	743	458
48K (4000)	**779**	**743**	**458**
64K (256)	939	903	532
64K (1500)	940	904	532
64K (4000)	**938**	**902**	**533**

Table 5-2 HBCE final storage requirements

5.2 Implementation & Performance

In this subsection we provide an analysis of the latencies of each block, the implementation cost of the reference design and its performance.

5.2.1 Latency Analysis

We calculate the minimum and the maximum number of clock cycles required by each hardware block in order to complete its operation. Many of the blocks have variable latencies which depend on the access patterns and the data stored in the data structures. Lookups are by far the most frequent and critical operations, and therefore, the device has been implemented in such a way that (in the average case) a lookup can be initiated at every clock cycle. However, even for lookups, in the cases of collisions in the MAC_TBL, the pipeline should stall, since more than one access to the external memory are needed. In Table 5-3 we present the latency for each HBCE block.

Block Name	Min Latency (clock cycles)	Max Latency (clock cycles)
HBCE_MCB	1	1
HBCE_INS	2	13
HBCE_LUP	1	10
HBCE_DEL	2	11
MAC_VID	1	1
MAC_HSH	1	1
MEM_HDLR	0	1
MEM_CTRL	1	1

Table 5-3 HBCE Blocks Latencies

All the blocks except of the ones performing the actual insert, lookup and delete, have single cycle latency in all cases. According to the number of observed collisions presented in subsection 5.1, the calculated latency for the very important lookup operation, is 4.98 clock cycles on average; the worst case latency is 13 clock cycles.

5.2.2 Hardware Cost Analysis

We have used VHDL to model the design and we have synthesized it using the Synopsys Design Compiler which is the most widely used synthesis tool. The underlying technology was UMC's 0.13μm CMOS one. Moreover, we used the Xilinx ISE tool to implement and port the design in an FPGA, as well, so as to measure its complexity in a low-cost environment.

Block	Area (mm^2)	Equivalent NAND Gates
Combinatorial	0,044	8482
Non-Combinatorial	0,054	10362
Total	**0,098**	**18844**

Table 5-4 Area estimations of HBCE

According to the synthesis tool the maximum working frequency of our design is 500Mhz. The silicon area covered by our design and the equivalent gate count is presented in Table 5-4.

FPGA's ISE tool reported that the maximum working frequency of the prototype design is 100 MHz in a state-of-the-art FPGA. Moreover, after certain optimizations for

removing redundant or replicated logic, the resources needed for the implementation of our design are those shown in Table 5-5. Those results have also been validated on real hardware since the design has been ported to a development board containing such an FPGA.

Resource	Resource count
Used 4 input LUTs	2371
Slice Flip Flops	1060

Table 5-5 FPGA resource allocation

5.2.3 Performance

The whole design is fully pipelined and thus the lookup performance of HBCE is based on the pipeline stalls which directly depend on the total number of memory accesses required to find a match in the tables (since we have just one memory). Both VLAN_TBL and PORT_TBL are directly mapped and therefore the associated FlowID can be found with a single access to the appropriate table. The MAC_TBL is the most critical table for the performance of HBCE, since collisions may occur and then we have to lookup sequentially all the colliding rules, triggering stalls in the pipeline. For every incoming MAC address the original vendor ID is replaced, in the first stage of the pipeline, with our internally assigned one. The VID_TBL is stored internally since it is quite small and therefore no external memory accesses are required in this first stage. The number of accesses/cycles required to resolve a MAC-address rule depends on the number of collisions that are associated with the corresponding entry in the MAC_TBL. According to the results presented in subsection 5.1, in the worst case there are 8 collisions when 64K rules are supported, while in the average case the number of collisions shrinks to 1.98. In Table 5-6 we present the summary of worst and average case pipeline stalls for each of our scenarios.

Active MAC Addresses	HBCE Average Case	HBCE Worst Case
32K	0.49	5
48K	0.73	6
64K	0.98	7

Table 5-6 HBCE total number of pipeline stalls

To calculate the network performance we have used two different memory modules : 200Mhz and 400Mhz synchronous SRAMs.

Active MAC Addresses	SRAM 200Mhz		SRAM 400Mhz	
	Average (Gbps)	Worst Case (Gbps)	Average (Gbps)	Worst Case (Gbps)
32K	**68,7**	17,1	**137,5**	34,2
48K	**59,2**	14,6	**118,4**	29,2
64K	**51,7**	12,8	**103,5**	25,6

Table 5-7 HBCE network performance

Since classification is performed for every incoming network packet, we calculate the throughput of our system based on the most conservative (worst-case) approach, by assuming that HBCE handles only minimum sized Ethernet packets (64 bytes). The summary of the supported link speeds is presented in Table 5-7 where the average throughput is based on the average pipeline stalls and the worst-case on the worst case stalls. This worst case is triggered when the lookups that should be performed, always encounter the maximum number of collisions.

The network performance presented in Table 5-7 allows HBCE to be used in a high speed Ethernet device that can support a large number of multi-gigabit ports. Those results demonstrate that our scheme can be used, for example, in a switch/concentrator supporting more than 100Gb/sec of aggregate throughput and 64K distinct classification rules.

6 Conclusions

The vast deployment of Multi-Gigabit Ethernet networks, and their use beyond the tight borders of LANs, motivated the development of QoS mechanisms in the MAC layer. Those schemes require classification of the Ethernet packets according to their MAC addresses, VLAN IDs or MAC port numbers. The proposed QoS mechanisms support many thousands of independent network flows; they handle separately and differently each of those flows. In order to address those needs, we have designed and implemented the Hash Based Classification Engine (HBCE) a classification module which uses an innovative hashing scheme and internal replacement of MAC Vendor IDs. HBCE can reach classification decisions at extremely high speeds, using significantly less memory than the existing solutions. The implementation of HBCE utilizes less than 0.1mm² of silicon area in a state-of-the-art CMOS technology, whereas it can support network rates higher than 100 Gb/sec, occupying less than half a Megabyte of Memory, and handling 64K distinct network rules.

References

[1] IEEE 802.1q Standard, *"Virtual Bridged Local Area Networks"*,

[2] Giovanni Pau et.al , *"A Cross-Layer Framework for Wireless LAN QoS Support"*, IEEE ITRE, August 11-13, 2003, Newark, New Jersey, USA

[3] IEEE 802.1p Standard, *"LAN Layer 2 QoS/CoS Protocol for Traffic Prioritization"*.

[4] *Enterprise Rack Mount Switches,* http://www.ncasia.com/rfq/24port_0303.cfm?rfq=Enterprise_24-port_rack-mount_switch

[5] R. A. Kempke and A. J. McAuley, *"Ternary CAM Memory Architecture and Methodology."* United States Patent 5,841,874, November 1998. Motorola, Inc.

[6] P. Gupta and N. McKeown, *"Algorithms for Packet Classification",* IEEE Network, March/April 2001, vol. 15, no. 2, pp 24-32.

[7] N. McKeown, B. Prabhakar, *"Lectures on Packet Switch Architectures II – Address Lookup and Classification",* Stanford University

[8] R. Jain, *"A Comparison of Hashing Schemes for Address Lookup in Computer Networks"*, IEEE Transactions on Communications, Vol. 40, No. 3, October 1992, pp. 1570-1573

[9] *VIA Networking Atlantic™ VT6510A Switch Controller*

[10] I. Papaefstathiou and V. Papaefstathiou*, "A Memory-Efficient, 100Gb/sec MAC Classification Engine",* 30th IEEE LCN 2005, November 2005, Sydney, Australia

[11] IEEE OUI and Company_id Assignments, http://standards.ieee.org/regauth/oui/index.shtml

[12] A. Nikologiannis, M. Katevenis: *"Efficient Per-Flow Queueing in DRAM at OC-192 Line Rate using Out-of-Order Execution Techniques"*, IEEE Int. Conf. on Communications (ICC'2001)*,* June 2001, Helsinki, Finland, pp. 2048-2052