An Integrated Hardware Solution for MAC Address Translation, MPLS, and Traffic Management in Access Networks

Harald Widiger and Stephan Kubisch and Dirk Timmermann
Institute of Applied Microelectronics and Computer Engineering
University of Rostock, 18051 Rostock, Germany
Telephone: ++49 (0)381 498 -3628
Email: {harald.widiger;stephan.kubisch;dirk.timmermann}@uni-rostock.de

Thomas Bahls
Siemens AG Communications
17489 Greifswald, Germany
Email: thomas.bahls@siemens.com

Abstract

Today, an increasing number of customers subscribes for a high bandwidth internet access. But not only communication speed is demanded. Quality-of-service moves more and more into the customers' focus. Both Carriers and Internet Service Providers (ISPs) have increasing requirements derived from new services they want to offer to their customers. We present a new hardware solution is presented, which can satisfy many of these upcoming demands. This solution is highly flexible and can be adapted to various applications. The MAC Address Translation - MPLS User Network Interface (MATMUNI) provides the functionality of MAC Address Translation (MAT), Multi Protocol Label Switching - User Network Interface (MPLS-UNI), and a Traffic Manager (TM). This way, a module implemented on a single FPGA offers a wide range of functionality in an access network for low costs and high flexibility. The selection of an FPGA as implementation target offers the possibility to adapt to future demands towards functionality. Moreover, the all functional elements work with wire speed. Only a negligible delay is inserted into the data path.

1 Introduction

Today, more and more customers subscribe for a high bandwidth internet access. But not only speed is demanded. Reliability, availability, quality-of service (QoS), and security move more and more into the customers focus [1]. As illustrated in Figure 1, the architecture of current access networks consists of various aggregation levels. The main aggregation points from the customers' side of the access network supporting Gbit-Ethernet are the linecards, the central nodes, and broadband access servers. Both central node and broadband access server support multiple Gbit-Ethernet streams and fiber optics. Every device in the data path has to analyze and process frame parameters such as source and destination addresses, QoS information, protocol types, and checksums to direct the data streams through the access network.

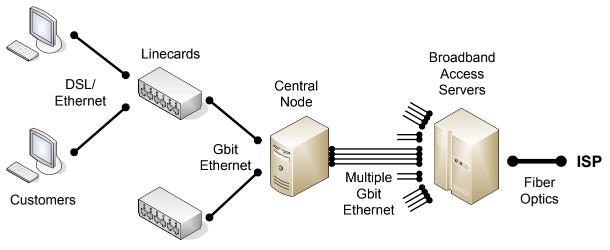

Figure 1. Current Access Network Architecture

Actually, home users favor the usage of Ethernet based DSL with intermediate bandwidths while business customers require connections with highest bandwidths and QoS. They even connect their own Local Area Network (LAN) directly to the carriers access network. Even the Plain Old Telephone System (POTS) may be integrated through Voice over IP (VoIP) into the access network. Thus, in the future, access networks have to deal not with a single customer connected to one line of the linecard. They must aggregate whole LANs at the linecards, as it is illustrated in Figure 2. This results in a huge number of users connected to the access network. Due to this high number of users, managing of address tables within the nodes and core switches becomes a challenging task. So called MAC address table explosions can occur [2]. Furthermore, as users are even able to manipulate their own addresses, which can result in duplicated addresses leading to severe network problems.

In addition, ISPs are selling more bandwidth than exists in assumption that not all customers are online at the same time using their committed bandwidth. This is commonly known as oversubscription. Thus, the customers share a common reservoir of bandwidth. But it might happen that during peak hours the customers claim what they paid for, because *”Internet users who pay a fixed fee have no incentive to limit their use of the network.”* [3]. In periods of decreased demand, valuable customers might be allowed to cause more traffic than they have subscribed for. Besides, business customers should be of higher priority than home users.

Moreover, customers demand QoS, diverse services, and traffic differentiation. In order to fulfill these various demands, it is necessary to enrich the different frames with additional information. The enrichment must take place within the access network, as data such as port information is available here only, but it is required at other locations of the network. The location of aggregation and service creation are different.

Software solutions targeting these problems can lead to an immense workload in the CPUs of the central nodes. That may prevent them from executing their primary tasks such as routing or switching. This offers vulnerabilities in the access architecture. Without flexible, resource aware, and economical management solutions, this is going to be a tremendous problem in the future. Obvious solutions utilizing Network Processors are too expensive and do not offer the performance that is required to be utilized in an access network environment. Other inexpensive hardware solutions cannot offer the functionality that is requested.

To cope with increasing demands for bandwidth and QoS and with scalability issues, the new solution, we propose in this paper, helps to manage the growing internet traffic in an elegant, flexible, cost-effective, and high-performance way.

Following, Sections 2 and 3 briefly introduce our hardware solutions for a MAC Address Translation (MAT) module, a Traffic Manager module, and a MPLS-UNI module. Section 4 presents the integration of all three solutions in one hardware module. Section 5 presents the advantages of our solutions and concludes this paper.

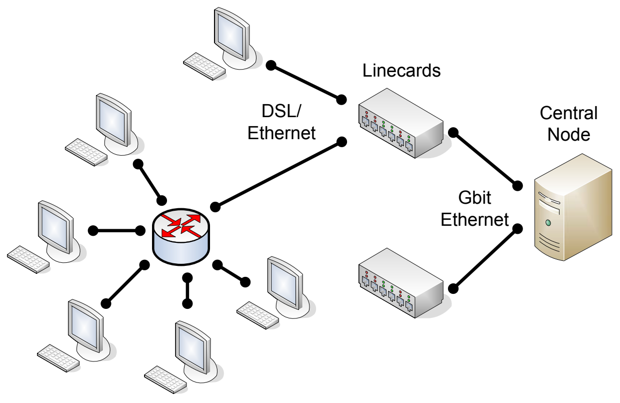

Figure 2. Customer Connections to Access Networks

2 Functional Modules

To cope with the aforementioned demands, three independent functional modules have been developed: a module for MAC Address Translation [4], a module for creation of an MPLS-UNI [5], and a module for Traffic Management [4]. The functionality is described in following subsections.

2.1 MAC Address Translation Module

MAT is a technique similar to NAT (Network Address Translation). But it is applied on layer 2 addresses instead of layer 3 addresses. MAT replaces the Customer MAC address (CMAC) of a data frame with a Provider MAC address (PMAC) and vice versa. In the upstream data path, the source MAC address is replaced, while in downstream, the destination MAC address is replaced. The replacement is performed based on information within the headers of each frame referred to as key. As sketched out in Figure 3, the key is extracted from each customer frame in the upstream data path. This key is processed and a truly distinct PMAC is returned to replace the CMAC. In the downstream data path the PMAC is replaced by the original CMAC which is stored in the modules address tables. Again, a key is processed to lookup the appropriate CMAC. Different relations between CMACs and a specific PMAC like 1:1 or n:1 are possible. That reduces the workload within the core network and addresses different aspects such as security (1:1) or scalability (n:1). The MAT functionality is placed behind the central nodes in the access network (see Figure 1). Therefore, the number of corresponding connected linecards and ports is limited. Thus, large address tables and address table explosions are eliminated.

Different MAC address encapsulation schemes exist. All schemes relief the workload of the core switches, because

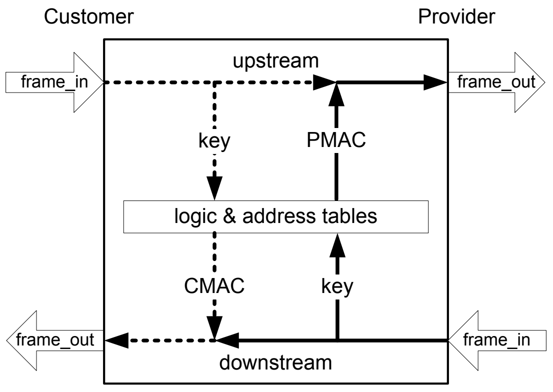

Figure 3. MAC Address Translation

switching decisions are based on the PMACs. MAT has similarities with MAC-in-MAC encapsulation (MiM) [6] and MAC address stacking (MAS) [2]. The difference is the replacement of the addresses instead of adding new header fields to the frame. Such becomes possible through sophisticated classification lookups. MiM adds a complete MAC header and MAS adds at least 12 Bytes (destination and source PMAC) which leads to larger frames, maybe exceeding the maximum frame size, and thereby leads to additional delays. Moreover, MAT as used here is feasible to be integrated in every Ethernet based network. It does not demand any extensions to existing switching hardware. It is fully transparent and conforms to the IEEE 802.3 standard.

2.2 Traffic Manager Module

The Traffic Manager module (TM) targets the problem of managing excessive load in the core network. In times of extremely heavy traffic, frames within the data path have to be discarded to resolve congestions in the network. It is desired not to discard frames randomly but to do so in a fair way. For this purpose, the TM module is used. The TMs functionality bases in priciple on the three color markers described in [7] and [8].

The TM is able to meter the traffic of each customer connected to the access network. Dependent on the actual data rates and two stored values, the committed information rate (CIR), and the burst information rate (BIR), the module marks every frame with colors ”green”, ”yellow” or ”red”. The mechanism works as follows: CIR and BIR represent the number of bytes the customer is allowed to transfer in a certain time. Two counters are reset to the CIR and BIR value if that time runs out. In the metering process, the length of each frame is subtracted from the counter values. Frames are marked green, if the customer does not exceed his CIR, meaning the corresponding counter is greater than zero. If he does and stays below his BIR, the frames are marked yellow. Otherwise, the frames are marked red.

With this color information coded into the frames, adjacent systems are able to discard red frames first. All yellow frames would be discarded next to resolve congestions before a green frame would need to be removed. This way, the green frames of all users are the last to be deleted assuring the CIR for all users as long as possible.

The color information can be coded into different parts of the frame. If the frames are MPLS labeled, the EXP-field of the label is used for the coding. Alternatively, the color can be coded into the TOS-field in the IP-header of a frame containing an IP packet. A third possibility is the utilization of the .1p Bits. The TM can operate both in up- and downstream direction. The module extracts a key from

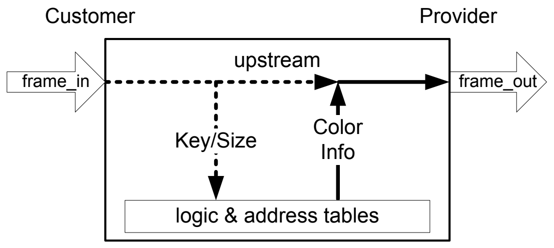

Figure 4. Traffic Management

the headers of each incoming data frame. The key identifies the customer. It can consist of different fields as already listed for the MATs key in Section II. The color information corresponding to the key is searched in a memory connected to the module. Depending on the retrieved information, the frame is colored. Furthermore, the color information in the memory is updated to perform the metering.

2.3 MPLS-UNI

As mentioned in Section I, it is needed to differentiate traffic in access networks, which is acomplishable by inserting information into every frame. To transport that information within the frames, MPLS label stacks should be used to carry all desired information.

MPLS is an encapsulation scheme. In an MPLS network, a label is assigned to each incoming packet. Packets are forwarded along a label switched path (LSP), where each label switched router (LSR) makes forwarding decisions based on the content of the label. But instead of using MPLS for switching and routing, it is intended to convey the aforementioned information in a standards-compliant and inter operable way. Therefore, no label edge router (LER) implementing the entire label distribution protocol (LDP) is required. The reduced functionality is realized with a simplified and cost-effective hardware solution as proposed in this paper. The modules’ purpose is to add an MPLS la-

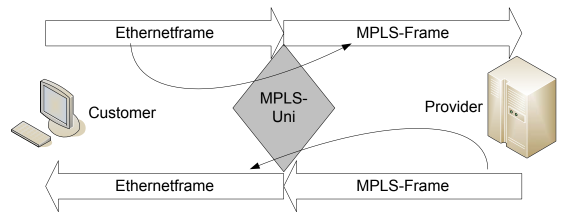

Figure 5. MPLS Labeling

bel stack to all incoming frames in upstream direction and to forward them to the providers' core networks. In downstream direction, the label stacks of all incoming frames are stripped off as shown in Figure 5. For setting up the frames with an MPLS label stack, the labeling scheme proposed by Martini et al. in [9] is used to conform to the MPLS hardware within the core network. This frame structure is given in Figure 6. Here, the frames are completely encapsulated. A new Ethernet header together with the created MPLS label stack is added in front of the original frame. The CRC checksum at the end of the frame is replaced by a new CRC value calculated over the whole new frame. The MPLS label stack consists of two different labels as pointed out in Figure 7. The values of both labels depend on the information corresponding to the key derived from the received frame. The inner label usually describes the virtual channel through the MPLS network. The outer label describes the actual path the frame has to take when passing the network to the corresponding egress point. However, for a certain application, at least 40 bits are available. Thus, a huge number of different services can be distinguished. VoIP data for example might use a much faster route than standard internet traffic. Even if the 40 bits should not suffice, more labels can be added to the stack.

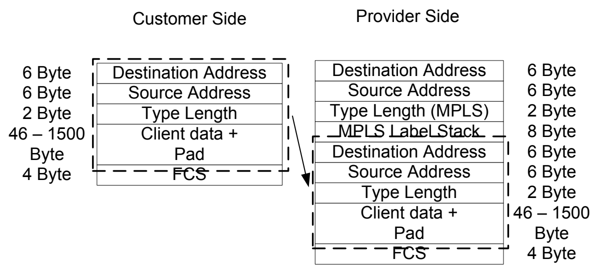

Figure 6. Frame Structures

In the following section the hardware architecture of our functional modules is described.

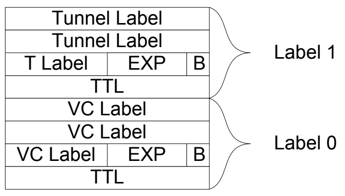

Figure 7. Added Label Stack

3 Hardware Architecture

All functional modules are written in VHDL (Very high speed integrated circuit Hardware Description Language). A reconfigurable silicon device will be used for hardware implementation. Thus, all modules are able to perform at wire speed. The FPGA is to be inserted between the central nodes and the broadband access servers in an access network (Figure 8). All modules base on the same principle

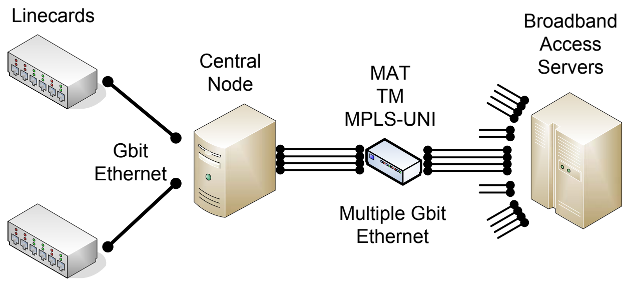

Figure 8. Access Network with MATMUNI

architecture. A block diagram of the core functionality is shown in Figure 9.

According to the constraints in actual access network environments, a throughput of four Gbit per second (Gbps) is demanded, reaching the hardware modules via four parallel Gbit Ethernet connections. That is why the data path has to be implemented as four parallel paths as well. Both in up- and downstream direction, a submodule extracts a key from the headers of each incoming frame and stores the frame in a buffer. The key identifies a customer, a group of customers, or a flow. Possible fields the key can consist of are source and destination MAC address, VLAN tags, ethertype, source and destination IP address, and the DSCP field in the IP header. Any combination of these fields can be configured. The configuration is done during synthesis to implement the desired functionality as efficient as possible. This way, the key parser differs in size depending on the desired key. In the minimal configuration, when the key parser contains only the byte of the DSCP field in the IP header, the implementation requires 336 slices. Parsing all seven pos-

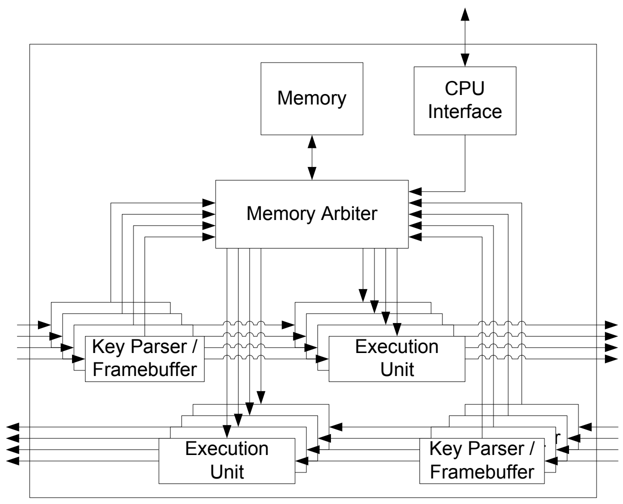

Figure 9. Architecture of the Functional Modules

switches, is avoided. This feature has no use in our application. In contrary to task switching in operating systems, no costs arise from changing the key parser unnecessarily.

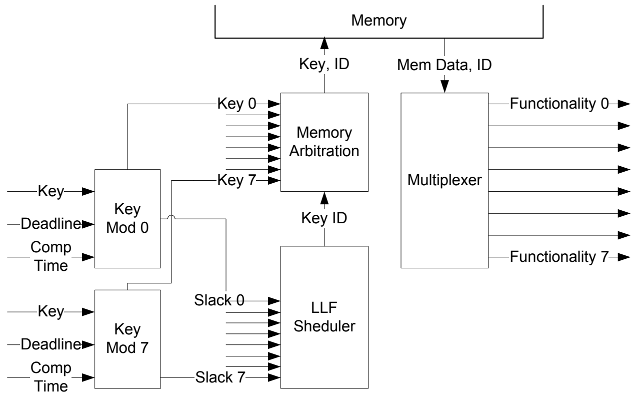

Figure 10. Architecture of the Memory Arbiter

sible fields requires 720 slices. In this paper, all presented results regarding implementation and speed refer to a Virtex4 V4FX20 FPGA with speed grade 10. However, the target device will be a Xilinx Virtex4 XCV4FX40 FPGA, because it provides sufficient resources to implement four Gbit channels and to extend the MATMUNI with additional functionality.

While the data frame is stored in the buffer, the extracted key is sent to a memory where the information referring to the key is stored. There are four key parsers in both up- and downstream direction. Thus, eight independent keys may request access to the memory at the same time. For that reason a memory arbiter (Figure 10) is required to schedule all memory requests. The memory arbiter decides, which of the competing keys gets access to the memory to search for the corresponding entry. This decision is made by applying the Least Laxity First (LLF) algorithm. The LLF, usually used for process scheduling in operating systems, assigns the memory access to the key with the smallest *slack*. In process scheduling, the smallest slack is computed as difference between two parameters of the process: the deadline to meet and the computation time required. In case of two equal slack values, the process with the smallest deadline is scheduled. In our application, the deadline derives from the size of the currently unused part of the frame buffer. The computation time derives from the number of frames that are already stored. Both deadline and computation time are presented as four bit values along with the key to the memory arbiter. The key with the smallest slack gets access to the memory. We considered using the Enhanced LLF (ELLF) algorithm as presented in [10]. ELLF enhances the LLF in the way that thrashing, i.e., unnecessary task

Any frame arriving via the four Gbit Ethernet links from any side must be classified. Thus, the possible number of memory lookups that can be performed in a given time is the bottleneck of the whole system. In order to decrease the number of memory lookups needed and therefore to increase lookup speed, all keys are stored in a *sorted* memory. This way, a binary search can be performed to find the information which corresponds to a key. This reduces the required number of memory accesses to a maximum of $\log_2(N)$ with N being the number of keys in the memory. The embedded MPLS architecture in [11] needs a linear search time, which is not sufficient when searching for information with a rate of incoming frames of 4 Gbps. Sorted memory entries of course complicate updates on the memory. Time consumption for insert and delete operations is increased, as after all operations the correct sorted structure of entries must remain. Inserting a new entry means finding the correct position and relocating all greater entries to the next higher address. Thus, insertion and deletion requires a maximum of $N + \log_2(N)$ memory accesses. However, these extra memory accesses can be invested. Insertions and deletions are usually only required if customers or networks connected to the access network change. Compared to normal search accesses to the memory, this insertions and deletions occur only very seldom.

When the information corresponding to the key is found, the functional module transfers the information to the execution unit. Then, the frame buffer is read out and the execution unit modifies the frame according to the modules' purpose. After its modification, the frame leaves the module either in direction to the core network or to the customer's edge. There are architectural elements in the functional module that are not pictured to keep track of the main functionality. Those elements have the purpose to configure the module and to transfer data from the data path to the CPU.

All three functional modules follow the above described architecture. However, the execution units and some parts of the other submodules differ slightly. The differences are detailed in the following subsections.

3.1 Traffic Manager

In the TM module (Figure 11), updates on the memories' content have to be performed regularily. Furthermore need the counter values for the green and the yellow counter to be updated after a frame has been color marked. That means,

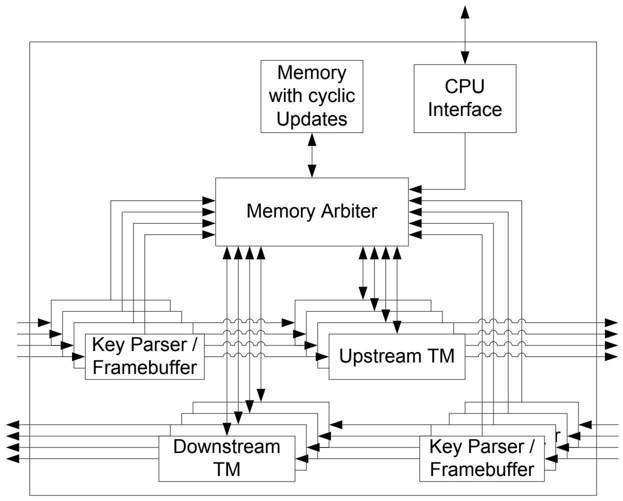

Figure 11. Traffic Manager Architecture

in addition to the key parsers, all execution units request access to the memory, too. This results in sixteen elements requesting access to the memory. In order to secure data integrity of the memory entries, counter updates must be executed with a higher priority than key lookups.

This way, it can be assured that counter information is updated before it is accessed again for a next lookup. Thus, CPU accesses to the memory have the highest priority, counter updates have a middle priority, and requests for key lookups have the lowest priority. Counter updates do not need to be scheduled in a special manner, as the next frame computation can occur not until all update operations have been finished. Thus, the implementation of a special scheduler as for the key lookups is not necessary.

3.2 MPLS-UNI

As mentioned in Section II, in downstream direction the MPLS-UNI has a very simple task. It unpacks all MPLS labeled frames arriving at the downstream ingress points. As

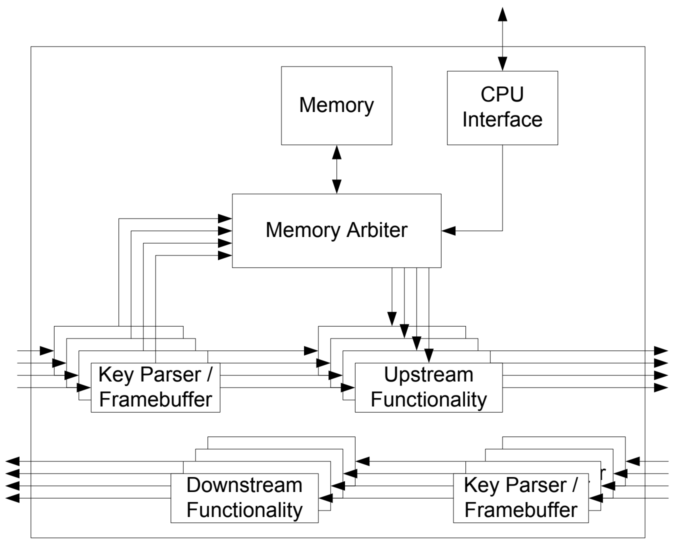

Figure 12. MPLS-UNI Architecture

the complete original frame is encapsulated by the MPLS frame, no additional information is needed to rebuild the original frame. The functional module strips off the Ethernet header and the MPLS label stack from the front of the frame. The truncated frame is sent to the customer's side of the module. In consequence, only the four key parsers in upstream direction compete for memory accesses in the MPLS-UNI module. The downstream key parsers do not need memory access.

4 An Integrated Solution - MATMUNI

The aforementioned functional elements can be combined to be used in a hardware that provides a broad set of functions. The interfaces of the data path are standardized and simple (Figure 13). It consists of an eight bit wide data signal, a Start of Frame (SoF) signal, and an End of Frame (EoF) signal. SoF indicates the first byte of a frame. The last byte of the frame is indicated by EoF. The transfer of a frame has to take place without interruption. This assures highest flexibility and interchange ability between the elements. Additional functional modules for further tasks can easily be integrated into the data path. The only requirement is that the interface conforms to the data flow, which is described above. With this interface, a pipelined data path providing much different functionality can easily be implemented.

But due to the mentioned similarities in the architectures of all three modules, an integrated solution was developed - MATMUNI. Figure 14 shows the MATMUNI system with

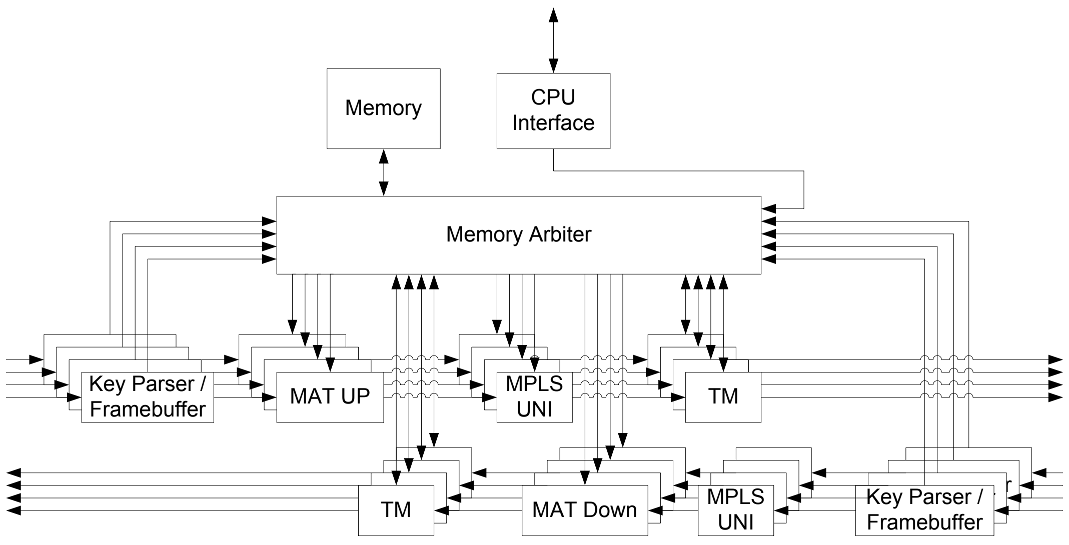

Figure 14. Architecture of the MATMUNI

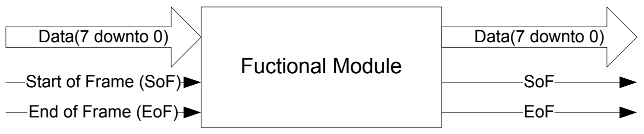

Figure 13. Data Path

execution units for MAT, MPLS-UNI, and TM. Functionalities common to all modules (key parser with frame buffer, memory arbiter) were chosen to be globally shared. Sharing those elements enhances efficiency of the implementation. The data parsers, e.g., require up to 2160 slices if implemented independent from each other. When they are integrated in one module just 720 slices are required at maximum. The union set of keys for all functional elements in up- and downstream direction is parsed and sent to the memory arbiter together. That means, merging the functional elements does not lead to performance penalties. The lookup can be done for all keys in parallel assuming independent memory blocks for every functional module. Independent memory blocks can be realized by using internal Block RAMs (BRAMS) of the FPGA, which is one way to implement the necessary memory. In case that the FPGAs internal memory is not sufficient, an external DRAM Chip must be used as memory device. This decision has great influence on capacity, cost, and performance of the system.

4.1 Internal Block RAM

In case internal BRAMs are used, it is possible to separate the memories of the different functional elements of the MATMUNI from each other. This has the great advantage, that all keys coming parallel from the upstream frame buffer (MAT, MPLS, TM) can be computed in parallel by three independent memories with search engines, too. Thus the lookup performance equals the performance of three non integrated, independent functional elements.

4.2 External DRAM

If external DRAM is to be used, there is no possibility of parallel memory lookups. Both merged and independent implementation solutions have to share the single existent memory interface. The three upstream keys have to be computed serially. One way of accelerating the memory lookups is to use the features of double data rate (DDR) memories. Due to transmitting two data words per clock cycle, DDR memory provides more memory lookups per second than internal BRAMs or standard DRAMs.

4.3 Implementation Results

As already mentioned, a first implementation was accomplished using a Xilinx Virtex 4 XCVFX20 FPGA. The amount of reconfigurable resources needed for the design considerably depends on the configuration of the MATMUNI. The MATMUNI can be configured in different ways. Firstly, all desired execution units (MAT, MPLS, TM in up- and downstream direction) must be selected. Secondly, the structure of the keys of each functional module must be defined. And thirdly, the number of Gbit channels used must be specified. These configurations can be made by changing constant values in a configuration VHDL

Table 1. Implementation Results

Module	Slices		Speed (MHz)
	min.	max.	
MAT	210		220
MPLS	220		160
TM	240		190
Memory Arbiter	282	1023	160
CPU Arbiter	600		320
Local Control Modules	160		180
Upstream Framebuffer	336	723	157
Downstream Framebuffer	336	723	157
MATMUNI	2300	4300	150

file. The complete functionality of the MATMUNI for one Gbit channel was synthesized. Depending on the type of the keys, the MATMUNI required 2300 slices when configured with a minimal key size. Configuring a maximum key size, the number of slices increased up to 4300. Additionally, some negligible glue logic was needed to synchronise incoming data into the MATMUNI. When the MATMUNI is configured with a typical key setup for all functional elements, it requires 3300 slices. With the same setup for all functional elements implemented in the stand alone version and connected to each other the hardware requirements nearly doubled. In that case 6000 slices were required for the implementation (MAT: 2100, MPLS-UNI: 1500, TM: 2400). Thus, the integrated solution saves nearly half of the hardware resources for an implementation. The module operated at a maximum speed of 150 MHz, although at least 125 MHz are sufficient to handle a data rate of 1Gbps per channel.

5 Conclusion

The presented hardware module provides a flexible and most of all cost-efficient solution in current and future access networks. It contains the functionality for layer 2 address translation, it expands MPLS networks into the access area, and it manages excessive network loads to ensure guaranteed QoS as long as possible. The module computes the data streams with wire speed. Thus, nearly no additional delay in the data path is generated. At a speed of 125 MHz, it can handle data rates of 4 Gbps in up- and downstream direction. Compared to the single implementations of all functionalities, the integrated solution provided by the MATMUNI saves nearly 50% of the hardware costs. Due to the fact that our solution is designed for reconfigurable hardware, the functional spectrum can be broadened and adapted to future tasks in the ever-changing and fast-living networking world.

This work is done in cooperation with Siemens AG Communications, Greifswald.

References

[1] R. Santitoro, “Metro Ethernet Services - A Technical Overview,” white paper, www.metroethernetforum.org, Tech. Rep., 2003.

[2] G. Chiruvolu, A. Ge, D. Elie-Dit-Cosaque, and M. Ali, “Issues and Approaches on Extending Ethernet Beyond LANs,” *IEEE Communications Magazine*, pp. 80–86, March 2004.

[3] L. W. McKnight and J. Boroumand, “Pricing Internet Services: Approaches and Challanges,” *IEEE Computer*, pp. 128–129, February 2000.

[4] S. Kubisch, H. Widiger, D. Duchow, T. Bahls, and D. Timmermann, “Wirespeed mac address translation and traffic management in access networks,” in *Proc. The World Telecommunications Congress 2006 (WTC06)*, May 2006.

[5] H. Widiger, S. Kubisch, D. Duchow, T. Bahls, and D. Timmermann, “A simplified, cost-effective mpls labeling architecture for access networks,” in *Proc. The World Telecommunications Congress 2006 (WTC06)*, May 2006.

[6] Nortel Networks, “Service Delivery Technologies for Metro Ethernet Networks. White Paper,” 2003.

[7] J. Heinanen, “A Single Rate Three Color Marker. RFC 2697,” September 1999.

[8] J. Heinanen, “A Two Rate Three Color Marker. RFC 2698,” September 1999.

[9] L. Martint et al, “Encapsulation Methods for Transport of Layer 2 Frames Over IP and MPLS Networks,” *internet draft*, February 2005.

[10] J. Hildebrandt, F. Golatowski, and D. Timmermann, “Scheduling Coprocessor for Enhanced Least-Laxity-First Scheduling in Hard Real-Time Systems,” 1999.

[11] R. Peterkin and D. Ionescu, “Embedded MPLS Architecture,” 19th IEEE IDPS, Tech. Rep., April 2005.