A Generic 10 Gbps Assembly Edge Node and Testbed for Frame Switching Networks

A. Mutter*, M. Köhn*, and M. Sund†
*University of Stuttgart, IKR, Pfaffenwaldring 47, 70569 Stuttgart, Germany
{mutter, koehn}@ikr.uni-stuttgart.de
†Alcatel-Lucent Deutschland AG, Bell Labs, Germany, matthias.sund@alcatel-lucent.de

***Abstract*—Packet assembly at the network edge is one solution to reduce the high packet rates in core network switches. For this, specialized edge nodes called Assembly Units are needed that assemble client packets into containers and vice versa.**

In this paper we present the detailed architecture and implementation of a generic Frame Assembly Unit for the Frame Switching architecture along with the testbed used for validation. Our design supports timer and threshold based assembly including packet fragmentation for fixed and variable size container frames at 10 Gbps per direction. For assembly and packet delineation we use the ITU-T Generic Framing Procedure. We report performance and implementation results for an overall design that operates with a 128 Bit data-path at 100 MHz on Xilinx Virtex4 FPGAs.

I. Introduction

Growth of data rates in packet core networks increases packet rates and thus the switching effort within core switches. This directly increases the complexity of these core switches. An approach to reduce the packet rates is to increase the size of the individual packets. A prominent solution for this is the assembly of smaller client layer packets into larger server layer containers at the network edge. The Frame Switching (FS) architecture [1] relies on this idea and shifts complexity and processing effort from the network core to the network edge, where rates are lower than in the core.

Fig. 1 shows a FS network. It consists of edge nodes called Frame Assembly Units (FAU) and core nodes called Frame Switches (FSW). At ingress, the FAU assembles client packets into containers and forwards them to the next FSW. The FSWs switch these containers along a preestablished path with support for relative quality of service (QoS) to the destination node. There, the egress FAU disassembles the container to individual packets and forwards them towards the destination client. As the containers are layer 2 frames, we use in the following the term frame for containers.

Frame Switching was originally proposed as an extension of

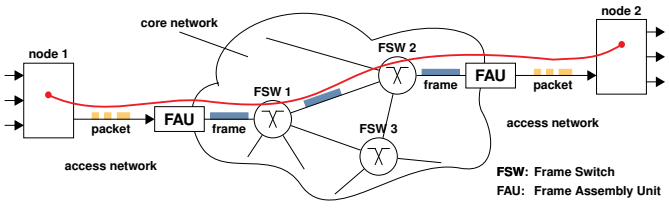


Fig. 1. Frame Switching network architecture

the Optical Transport Network (OTN). However the concept of packet assembly at the network edge is also applicable to Ethernet. Frames of both network architectures differ with respect to their size restrictions. While OTN uses fixed-size G.709 frames Ethernet frame sizes are variable between a well defined upper and lower bound. Due to the size limitation in Ethernet client packets face fragmentation to achieve maximal throughput. Using fixed size OTN frames client packet fragmentation is mandatory to fill the frames efficiently. Further, fixed size frames must be padded in low load situations.

In literature, several architectures and implementations of assembly nodes as well as testbeds have been presented for Optical Burst Switching (OBS) networks [2]–[6]. They all support only variable length burst assembly and thus do not consider packet fragmentation impact as well as padding. The works in [2], [3] mainly focus on some key unit technologies, such as optical switches. In [4], [5] the authors describe the node architecture in more detail but give no information about how assembly and disassembly work. The authors in [6] use a standard network processor to implement an assembly node and describe in detail the assembly process. Nevertheless, scalability and throughput are limited due to the software implementation.

For Frame Switching networks to the best of our knowledge only [7] describes an assembly node architecture. The authors present the nodes' ingress direction able to assemble packets into fixed sized frames. Assembly is done timer and threshold based at a line rate of 10 Gbps. The authors use Generic Framing Procedure (GFP) for user packet encapsulation and G.709 as frame format. Nevertheless, the work lacks the egress direction with disassembly part and neglects fragmentation.

In this paper we present the architecture of a generic bidirectional FAU including its implementation (prototype) and a complete testbed for validation. Our FAU architecture is highly modular and eases adaptation to any packet oriented technology. We describe its ingress and egress direction and argue our design decisions. Our FAU prototype supports fixed and variable frame sizes including fragmentation and padding. It assembles based on a combined timer and threshold based assembly strategy at a throughput of 10 Gbps per direction. We describe how we solved assembly and disassembly by the use of ITU-T Generic Framing Procedure. Our testbed enables further performance studies and research on how real applications are affected by packet assembly.

This paper is organized as follows: section II describes a generic assembly node's functional architecture. Section III shows the testbed's network scenario which was used to validate the FAU. Section IV argues the architectural design decisions made and describes the FAU's prototype architecture in detail. Finally, section V presents implementation details and performance results.

II. Functional architecture of a generic assembly unit

The key element in a frame switched network is the Frame Assembly Unit (FAU). It assembles packets to frames in ingress direction, i. e. from access to core, and disassembles frames to packets in egress direction, i. e. from core to access. This chapter introduces the functional architecture for a generic FAU.

Fig. 2(a) depicts the functional architecture for ingress direction. From left to right, incoming packets are classified and assigned to a Forwarding Equivalent Class (FEC).

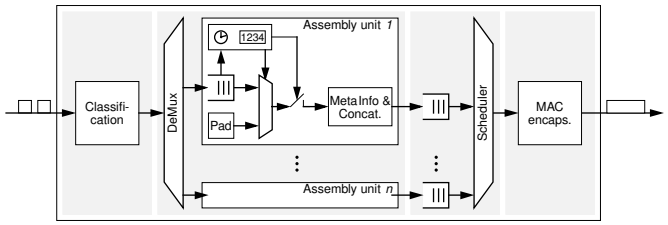
The assembly stage assembles packets of the same FEC to frames. Therefore the FIFO in the corresponding assembly unit collects arriving packet data. The control block monitors the FIFO fill level and triggers a frame generation when the size threshold is reached. A timer is started upon packet arrival into an empty FIFO to avoid starvation. Upon timeout a frame is generated. In case of fixed size frames the assembly stage fragments packets to completely fill the frames. It appends padding if the amount of collected packet data is below the minimum frame size. Before concatenation to one continuous data block meta information is added to enable packet delineation in egress direction.

Frames which are ready to be sent are buffered in case of congestion and scheduled according to their Class of Service (CoS). The MAC encapsulation stage finalizes the frame for transmission by adding headers and trailers.

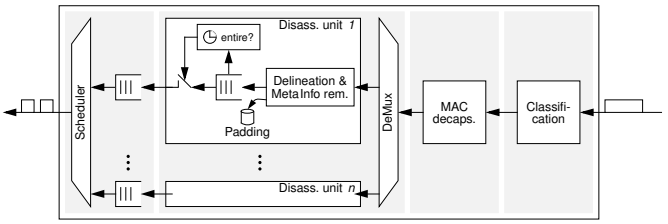
Fig. 2(b) shows the functional architecture of the egress direction of a generic FAU. From right to left, incoming frames are classified and assigned to an FEC. The MAC decapsulation stage removes the frame overhead.

The disassembly stage delineates packets with help of the meta information added during assembly. It also drops the meta information as well as padding. In case of fixed size frames the last data in a frame may be a packet fragment. The FIFO queue stores packets and packet fragments. The control block monitors the FIFO and if it contains an entire packet it triggers its forwarding. Similar to ingress direction a buffer stage and a scheduling stage take care of packet transmission according to their CoS.

For mapping a client packet data stream into a server constant bit rate (CBR) stream the ITU-T has standardized the generic framing procedure (GFP) [8], [9]. GFP encapsulates each packet by adding meta information (the GFP core and payload headers) that allows delineation at the destination. Furthermore, if no packet data is available GFP idle frames are inserted for padding. Finally it supports fragmentation of packets and distribution into multiple server frames. However



(a) ingress direction



(b) egress direction

Fig. 2. Functional archtitecture of a FAU

the server layer is not CBR in our case, we use GFP for padding and delineation.

III. Testbed network scenario

We set up a testbed for the complete Frame Switching architecture to validate the function and realizability of the network architecture as well as of the generic FAU architecture. This testbed comprises both data and control plane. In this paper, we focus on the data plane part only, which is depicted in Fig. 3.

In our testbed we use 10 Gigabit Ethernet equipment for both, access as well as core network. Quasi-standard *Ethernet jumbo frames* of 9 Kbyte builds our container frame within the core. We interconnect three access networks transparently by transporting the entire Ethernet layer through the FS network.

The testbed consists of three FAUs, one core switch and three access switches. The access switches A, B and C aggregate client traffic to one 10 Gigabit Ethernet link and map client packets to one Virtual LAN (VLAN, [10]) per FEC. The latter simplifies classification within the FAU and decouples FECs from the attached clients' MAC addresses. Upon change of a client the classification criteria need not to be updated.

The FAU assembles packets belonging to one FEC into frames and forwards them into the core network. As FS is connection oriented while Ethernet is connection less we have to emulate a connection in the core. For this, we configure for each bidirectional connection one VLAN. This VLAN contains only those two ports of the connections data path. This enables also QoS differentiation in the core switch by using the VLAN-priority field in the VLAN header.

The remaining part of the paper focuses on the architecture and realization of the FAU prototype.

IV. Frame Assembly Unit

This section describes the architecture of the FAU prototype. We first argue major design decisions that realize the requirements on functionality and throughput. Then we present the architecture in detail.

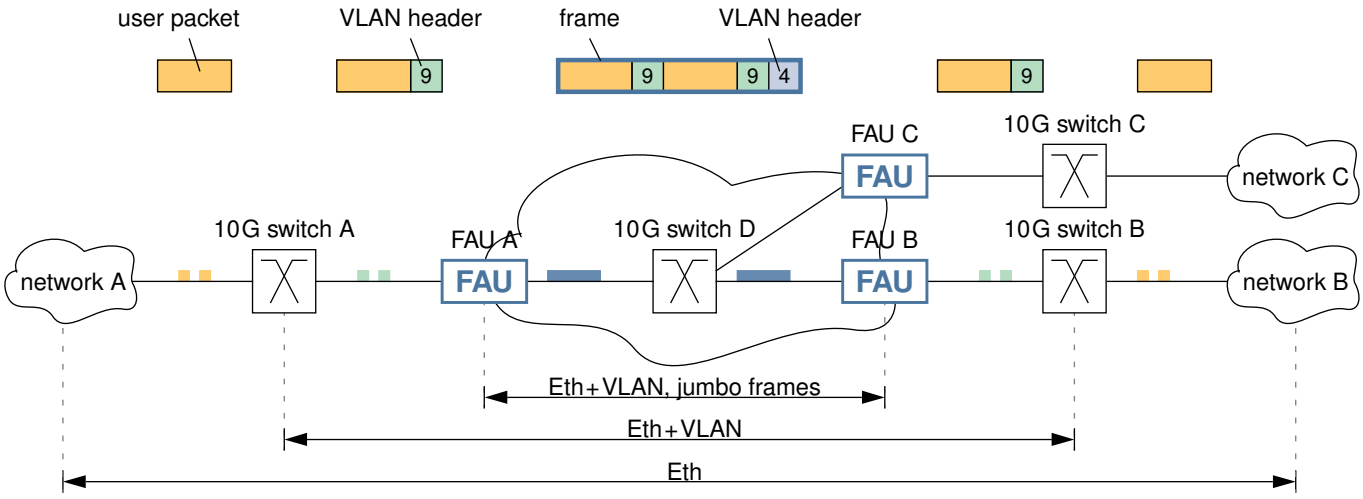


Fig. 3. Testbed network scenario

A. Design decisions

First we have to decide between a hardwired implementation versus an implementation based on programmable processors for the function blocks. A hardwired implementation is always faster because it is specialized and usually works in a highly parallel manner. In contrast, programmable processors are more flexible. The FAU processes header and payload of all packets. This is due to GFP's requirement for scrambling and descrambling entire packets during encapsulation and during decapsulation respectively. Furthermore a byte by byte inspection of incoming frames is necessary during packet delineation. To handle the required data rate of 10 Gbps we use hardwired implementations for all function blocks.

Second is the organization of function blocks as a pipeline or as a parallel structure, or even a combination. A pipeline with hardwired stages enables decomposition of functions into small steps to increase frequency and throughput as well as modularity. As the FAU processes all data equally and in a fixed sequence per direction, we chose a pipeline structure for our architecture.

To increase throughput we could use several parallel processing units per pipeline stage or even use several parallel pipelines. In several processing units (e. g. GFP), the state before processing a packet or a frame depends on the final state of the previous one of the same FEC. This means that data belonging to the same FEC cannot be processed by two parallel units at same time. As we assume that every FEC may use the full line rate, even if only temporary, every processing unit has to support line rate. Therefore it is not reasonable to parallelize processing units.

To realize data processing with per FEC state two options arise. First is to include one processing unit instance per FEC into the stage – that obviously does not scale. Second is to use only one unit per stage and extend it by the ability to change the state according to the processed FEC with help of a state memory. For this prototype, we apply both concepts. For complex stages we use the second possibility to save resources. But for simplicity reasons we use parallel units to implement assembly and disassembly stage as only a small number of FECs is required. This also enables dedicated buffers in these stages which in turn simplifies the implementation.

Third is the buffering concept in assembly and disassembly stages. In ingress direction, buffers can either store individual packets which are concatenated into frames afterwards or the packets are concatenated first and buffers store these larger frames. First option allows a fine-grained modularization because packet concatenation to a frame as well as the logical assembly are separated. The second alternative leads to a more efficient use of memory bandwidth at the expense of additional complexity. The egress direction is identical to its ingress counterpart except for granularity: buffers can either store packet fragments which are concatenated into packets afterwards or packet fragments are concatenated first and buffers store packets.

For real data traffic the increased memory bandwidth is negligible. But as the separation of functionalities reduces implementation complexity, we chose for both directions data buffers that store packets and packet fragments, respectively.

Finally, nowadays line rates of e. g. 10 Gbps are more then an order of magnitude higher than achievable processing frequencies on an FPGA or ASIC. Therefore, processing has to be done highly parallel and on-chip data buses have to be wide to provide the required throughput.

But as packet lengths have byte granularity, a wide bus also has two main drawbacks. First is the alignment problem. If packet lengths are changed, packets are concatenated or frames delineated, data must be realigned within the bus. For these operations, an increase in bus width leads at least to a linear increase of hardware effort. Second, the bus efficiency decreases for wider buses. As each bus word always belongs to only one packet, its last word is usually filled only partially. In the worst case concerning data throughput the last word contains only one valid data byte. This is also called the 65 byte problem. Potential bus throughput has to be over-dimensioned accordingly to cope with this problem.

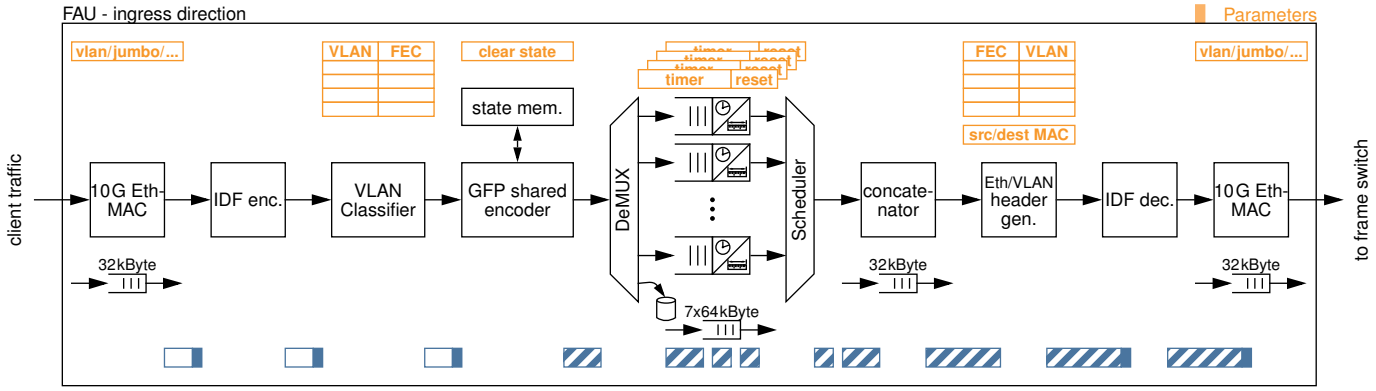


Fig. 4. Architecture of FAU prototype: ingress direction

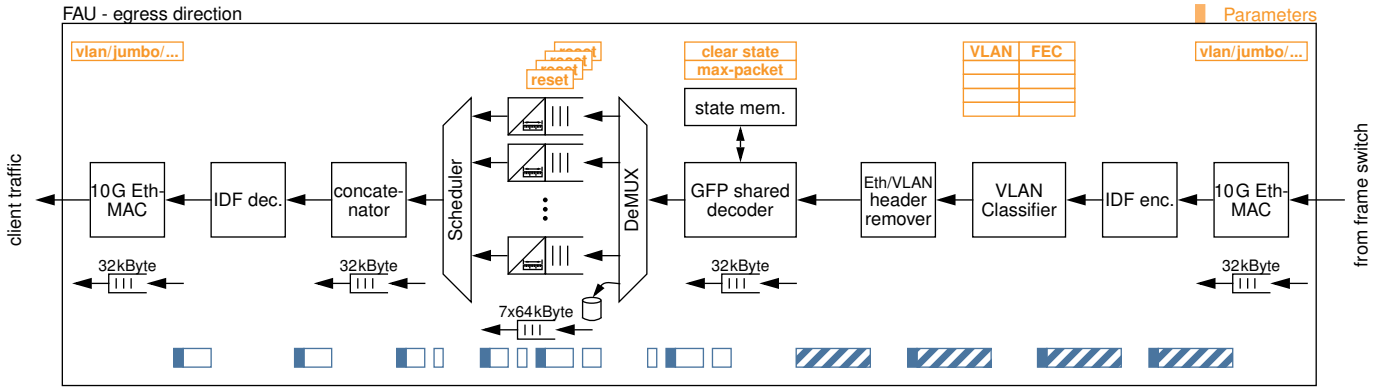


Fig. 5. Architecture of FAU prototype: egress direction

B. Prototype architecture

Fig. 4 and Fig. 5 depict the implemented architecture of the FAU in ingress and egress direction respectively. Both Figures show on top available configurable parameters during operation, in the middle pipeline stages and at the bottom the processing state of the packets/frames. Furthermore, a FIFO symbol marks stages with FIFO buffer and their sizes. We describe both directions along the data processing path.

First two stages receive incoming packets, buffer them temporarily and convert them to the internal data format (IDF). Therefore they adapt the data width to the internal bus width and add an internal header.

The VLAN classifier stage assigns each packet to an FEC and stores this information inside the internal header. Then packets are GFP encoded. Therefore, the GFP encoder adds GFP core and payload headers and scrambles the data. We positioned the GFP encoder before the assembly unit as with this the packets have already their final length during assembly. This simplifies size calculation inside the assembly unit.

Afterwards, the GFP encoded packets are demultiplexed to the according assembly unit while packets not assigned to an FEC are dropped here. Each FEC has its assembly unit with a dedicated buffer that is also responsible for the timeout and size control. On timeout or on reaching a defined size threshold the assembly unit signals to the scheduler that a frame is ready to be sent. On request of the scheduler it transmits GFP encoded packets or fragments of a packet to the concatenator stage to build a fixed or variable length frame. In case of fixed size frames, it additionally generates GFP idle frames as padding if necessary and transmits them to the concatenator. If more than one assembly unit signals ready to send, the scheduler assigns the outgoing link in Round Robin manner to the assembly units.

The concatenator stage removes IDF headers, aligns data and creates a continuous data block which will build the frame's payload. This data block is forwarded to the Eth/VLAN header generator stage which prepends an Ethernet and a VLAN header according to the frame's FEC to build a valid Ethernet jumbo frame.

Finally, last two stages remove the IDF header, buffer the frame and transmit it with line rate. Here again data width is adapted to the data width used by the outgoing interface.

In egress direction the first two stages receive incoming jumbo frames, buffer them temporarily and convert them into IDF similar to ingress direction.

The VLAN classifier stage then assigns each frame to an FEC and stores this inside the internal header. The Eth/VLAN header remover stage cuts all Ethernet and VLAN headers. The remaining data block is forwarded to the GFP decoder stage. This stage delineates GFP encoded packets, removes GFP headers and drops GFP idle frames. It forwards resulting packets and packet fragments to the disassembly stage.

After GFP decoding, packets and packet fragments are buffered in the disassembly stage where each FEC has its

disassembly unit with dedicated buffer. As in ingress direction, packets not assigned to an FEC are dropped here. As soon as a disassembly unit contains an entire packet it signals the scheduler that a packet is ready to be sent. On request of the scheduler it transmits all fragments of one packet to the concatenator stage.

The concatenator stage removes IDF headers, aligns packet fragments and creates a complete packet. This packet is forwarded to the last stages where, like in the ingress direction, the IDF header is removed, the packet is buffered and then transmitted with line rate.

V. Realization

A. Evaluation Board

We use an AdvancedTCA (Advanced Telecom Computing Architecture) compliant printed circuit board for evaluation (Fig. 6). Two Xilinx Virtex-4 FX100 90nm FPGAs (package FF1517) build its core.

One FPGA implements a GMII (Gigabit Media Independent Interface) connected to a MARVEL 88E1111 tri-mode Ethernet transceiver device. Each FPGA is connected to a fully protocol agnostic transceiver device (Vitesse VSC8479) via two source synchronous, unidirectional LVDS interfaces effectively implementing a XSBI (10 Gigabit Sixteen Bit Interface) to access XFP (10 Gigabit Small Form Factor Pluggable) optic modules. XSBI signal traces are length matched to prevent signal alignment issues at FPGA I/Os and to avoid tedious signal delay adjustment tasks within the FPGAs. Vitesse transceivers' operating speed is 644.53125 MHz resulting in the required 10GBASE-R line rate of 10.3125 Gbps. They are connected over short (2.5 cm) traces to their attached XFP modules for signal integrity reasons.

To interconnect the FPGAs, three types of interfaces are implemented: First type is a clock and control interface with low pin count. Second type is a parallel interface that consists of 32 data and 2 clock signals. This is three times available. The third interface is a high speed serial interface that utilizes one MGT (Multi Gigabit Transceiver) to implement a channel for data rates in the range of 10 Gbps. Four of them are available.

B. Implementation

We designed the FAU prototype architecture using VHDL and Xilinx ISE 10.1.02 for synthesis and place&route. The internal bus as well as all processing units have a width of 128 Bit. The whole design except the external interfaces operates at 100 MHz. The gross throughput of the FAU is 12.8 Gbps per direction. All stages in the architecture are clocked and can store one or more bus-words in registers. As 10 Gigabit Ethernet MAC we use an Intellectual Property core from Xilinx in Version 8.5 while all other functionality was implemented by ourselves.

As both FPGAs on the evaluation board are connected exclusively to one 10 Gigabit Ethernet interface the implementation has to be mapped to both FPGAs. Fig. 7 depicts this mapping. The ingress direction is on FGPA0 and the egress direction on FPGA1. With this mapping, the 10 Gigabit Ethernet interfaces can not be distinguished into core and access interfaces. Instead, both are half access and half core interfaces. Due to this, optical fibers connecting core and access Ethernet switches have to be split up in RX and TX direction such that they can be correctly connect to the board, as depicted in Fig. 7.

FPGA1 additionally contains the FPGA Management System (FMS) which serves as control plane interface. Via FMS which uses the 1 Gigabit Ethernet interface parameters can be configured for both ingress and egress direction. Therefore, FMS uses two parallel inter-FPGA links. The FAU's current configuration as well as a lot of statistics can be set and monitored during operation utilizing a control PC.

For modularity and flexibility reasons all stages in the pipelined architecture have uniform bus interfaces.

All but the classifier stage change the length of transported data. This makes processing time nondeterministic and necessitates a flow control. To handle data flow control between stages we use a bus protocol which differentiates between two kinds of stages: stages with FIFO buffers and stages without. Between two stages with FIFO buffer we can transfer a data block without interruption. Therefore the sender asks the receiver if a transaction is ongoing and for the amount of free memory. The sender can start to send data when no transaction is ongoing and there is enough free memory. A transaction ends as soon as the data block is received completely. Stages without FIFO buffer cannot interrupt a transaction and must receive one data word per clock cycle during a transaction. An additional register in these stages avoids overflow in case of word wrap around.

Inside the GFP encoder and decoder stages, we implemented Frame mapped GFP (GFP-F) [8] standard compliant. To process data from different FECs, both have a state memory to store the processing state and support a fast state switch. In contrast to [11] we also realized payload processing and this also for several FECs.

We implemented all FIFO buffers in the architecture with on-chip dual port memory. We dimensioned them twice as large as the largest packet/frame for simplicity reasons to enable blocking free operation. Assembly and disassembly units are exceptions. They were dimensioned for zero packet loss during assembly and disassembly as follows:

With respect to buffer occupancy the worst case scenario for an assembly unit is, when all assembly units contain enough data to send a frame and all the following data is destined to the assembly unit that processes the FEC with lowest priority. To avoid packet loss, this assembly unit needs the number of FECs times the maximal frame size of memory. This is here $\lceil 7 \cdot 9\text{Kbyte} \rceil = 64\text{Kbyte}$. The number of supported FECs in ingress direction is limited by the total available on-chip memory. For disassembly same calculation can be made when using the maximal packet size instead the frame size. As we do not limit the user packets size to 1518 byte we use the same dimensioning in both directions.

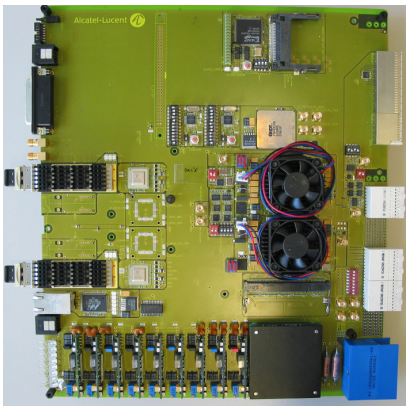


Fig. 6. Evaluation Board for FAU prototype implementation

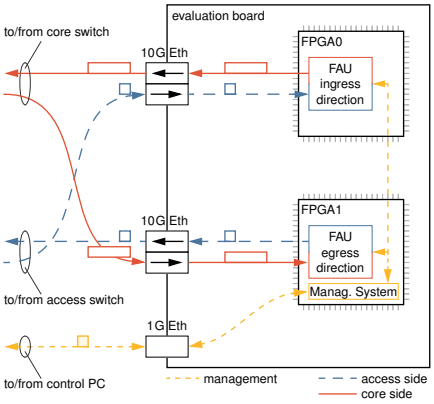


Fig. 7. Mapping FAU functionality to evaluation board with two FPGAs

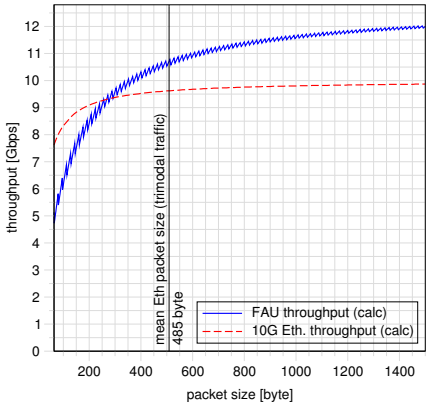


Fig. 8. Throughput of FAU compared to maximal 10G Ethernet throughput

C. Numerical Results

The implementation of the ingress direction uses 35 % flip flops and 52 % look up tables available in FPGA0. The egress direction including FMS uses 29 % flip flops and 42 % look up tables available in FPGA1. These numbers also include approx. 50 statistic counters per FPGA with 32 to 64 Bit width. The maximum achieved Frequency is 105 MHz.

The design was validated successfully using an on-chip pseudo-random data generator as well as within the described testbed scenario. For the first, ingress and egress direction were put back to back on one FPGA including two pseudo-random data generators, one generating, one verifying packets.

Fig. 8 shows the net throughput of the FAU as a function of packet size compared to the maximal net throughput of 10 Gigabit Ethernet. For the minimal packet size of 60 byte (minimal Ethernet frame without CRC) the FAU reaches a throughput of 5 Gbps. This is due to the unused clock cycles between two transactions. For large packet sizes the FAU throughput converges to the gross throughput of 12.8 Gbps.

The FAU throughput curve has a saw tooth shape due to the internal bus width. The fill ratio of a packets last word depends on its length, what impacts internal bus efficiency. With increasing packet lengths the relative weight of one word decreases, which leads to smaller saw teeth. An increase in bus width would lead to overall larger saw teeth.

VI. Conclusion

In this paper, we presented the architecture and implementation of the ingress and egress direction of a fully operational Frame Assembly Unit. We propose a pipelined architecture with uniform stage interfaces maximizing throughput. The architecture is modular to ease design adaptation to any packet oriented transport technology. We described the design in VHDL with a 128 Bit wide data-path to handle 10 Gbps throughput per direction.

Our architecture supports packet fragmentation for a 100 % container frame fill ratio and Generic Framing Procedure for packet delineation. The prototype is able to handle 7 connections per direction.

We validated the design within a testbed using Ethernet packets as client traffic and Ethernet jumbo frames as containers. The presented design proves that a complete assembly node is viable on a standard FPGA device. Our testbed enables further performance studies and research on how real applications are affected by packet assembly.

Acknowledgement

This research was done in cooperation with Alcatel Lucent Bell Labs Germany, Stuttgart. The authors would like to thank Sebastian Gunreben for his valuable contributions and all discussions.

References

[1] H. C. Leligou, et al., "Hybrid burst/packet switching architectures from IP Nobel," in *Optical Transmission Systems and Equipment for Networking V*, vol. 6388, no. 1. SPIE, 2006, p. 63880C.

[2] R. Nejabati, D. Klonidis, D. Simeonidou, and M. O'Mahony, "Demonstration of user-controlled network interface for subwavelength bandwidth-on-demand services," in *Optical Fiber Communication Conference*, vol. 3, 6-11 March 2005.

[3] F. Masetti, et al., "Design and implementation of a multi-terabit optical burst/packet router prototype," in *Optical Fiber Communication Conference and Exhibit*, 17-22 Mar 2002, pp. FD1–1–FD1–3.

[4] Y. Sun, et al., "Design and implementation of an optical burst-switched network testbed," *IEEE Communications Magazine*, vol. 43, no. 11, pp. S48–S55, Nov. 2005.

[5] P. Pradie, E. Rodriguez, and A. Agusti-Torra, "Optical burst switching network data path design and implementation," in *Proceedings of EUNICE 2006*, 2006.

[6] J. Koegel, S. Hauger, S. Junghans, M. Koehn, M. Necker, and S. Stanchina, "Design and evaluation of a burst assembly unit for optical burst switching on a network processor," in *Proceedings of EUNICE 2005*, 2005, pp. 3–16.

[7] G. Kornaros, W. Lautenschlaeger, M. Sund, and H.-C. Leligou, "Architecture and implementation of a frame aggregation unit for optical frame-based switching," *International Conference on FPL 2008*, pp. 639–642, Sept. 2008.

[8] ITU, "Rec. G.7041/Y.1303: Generic Framing Procedure (GFP)," International Telecommunication Union, ITU-T, Dec. 2001.

[9] E. Hernandez-Valencia, et al., "The generic framing procedure (GFP): an overview," *IEEE Communications Magazine*, vol. 40, no. 5, pp. 63–71, May 2002.

[10] R. Seifert, *The All-New Switch Book*. Wiley, 2008.

[11] C. Toal and S. Sezer, "A 10 Gbps GFP frame delineation circuit with single bit error correction on an FPGA," *Proceedings of the AICT/SAPIR/ELETE 2005.*, pp. 357–362, July 2005.