Optimized Ethernet transmission of acquired data from FPGA to embedded system

Wojciech M. Zabołotny[a]

[a]Institute of Electronic Systems, Warsaw University of Technology, ul. Nowowiejska 15/19, 00-665 Warszawa, Poland

ABSTRACT

This paper presents a simple system consisting of the FPGA core, network protocol and Linux kernel driver, aimed on efficient transmission of acquired data from the low resources FPGA equipped with Ethernet PHY to the embedded system, responsible for preprocessing of those data and sending them further via standard network links.

The system has been optimized regarding the memory and logic consumption in the FPGA. Implementation based on the Layer 3 protocol allows to minimize latency of the packet acknowledge, which results in reduction of memory requirements on the FPGA side. The driver code has been optimized to avoid unnecessary copying of data between buffers in memory, allowing the user application to access received data via memory mapped buffer. The system has been successfully tested in real hardware. Sources of the whole system are published and freely available under Open Source licences (partially under GPL, partially under BSD and partially as public domain).

Keywords: FPGA, Ethernet, Embedded systems, Data acquisition system, Layer 3 protocol, data transmission, open source

1. INTRODUCTION

Transmission and concentration of data from multiple input channels is a significant problem in modern measurement systems. The cost of the data transmission subsystem may be a significant part of the total cost of the system. Additionally this subsystem is essential for reliable operation of the whole measurement system, so it is important that it is based on well tested hardware components, and for good maintainability it is also desirable, that those components are widely available.

Considering the above factors, the Ethernet network seems to be a good choice to build such a data transmission and concentration subsystem.

In the typical measurement system, the sensors and detectors are connected to the Front End Boards (FEBs). These boards receive input signals, convert them to the digital form, typically supplement with the time stamps and send them to the system components responsible for further processing of concentrated data, which typically are the computer systems.

The FEB boards are often based on the Field Programmable Arrays (FPGA) chips, due to their good performance in parallel data processing, easy interfacing with different interfaces used in the input sensors and converters, and easy generation of precise timing relationships between different signals, needed to provide deterministic and jitter-free sampling of input signals.

Another important property of the FPGA chips is their flexibility, and possibility to correct possible errors by simple modification of the chip configuration (firmware), which can be done without hardware modification.

As the FEB boards are typically one of most numerous components of the measurement system, they should be price optimized. It means, that they should use possibly cheap FPGAs with minimized number of external components. To combine advantages of both: Ethernet based data transmission and FEB boards based on inexpensive FPGA chips, it is necessary to implement a method for reliable data transmission from small FPGA to the computer system via Ethernet link.

Further author information: (Send correspondence to W.M.Z.)
W.M.Z.: E-mail: wzab@ise.pw.edu.pl, Telephone: +48 22 234 7717

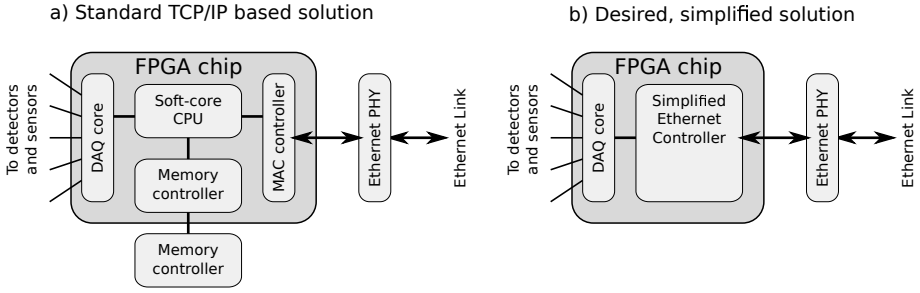

Figure 1. Two possible methods to connect the FPGA chip to the Ethernet network. The first one is based on soft-core CPU with MAC controller and external memory for data buffering. The second one uses the simplified Ethernet Controller and minimizes amount of external components.

2. USE OF FPGA WITH ETHERNET INTERFACES

The FPGA may be easily connected to the Ethernet PHY, allowing to transmit and receive Ethernet packets. Many FPGA development kits, e.g. the Spartan 3E Starter Kit,[1] Spartan 6 based SP601[2] board and also Spartan 6 based Digilent Atlys[3] board, all used in this research, offer such configuration.

There is however one significant problem, associated with the Ethernet transmission. The Ethernet does not assure reliable transfer of data. The bit error rate in the physical link may be up to 10^{-8} in the 10Base-T[4] Ethernet, and up to 10^{-10} in the 100Base-T[5] and 1000Base-T[6] Ethernet. It is also possible, that the receiving computer may drop some received packets due to insufficient resources to process them in a timely manner. Additionally, in some links there is a risk of packet corruption due to collisions, however this problem may be avoided by using only full duplex links in the data transmission system.

The inherent unreliability of Ethernet transmission is compensated in the higher layers of the network protocols. The most popular solution providing reliable transmission of data through unreliable network connection is the TCP/IP protocol with reliability achieved by implementation of the acknowledge/retransmission mechanism. There are some TCP/IP implementations available for FPGA chips,[7–9] but they require a lot of resources to implement the soft-core CPU or complex state machines and a lot of memory to buffer the data (see Figure 1 a).

What we need, is a solution providing reliable transfer of data via Ethernet link, even from the small FPGA chip without implementing soft-core CPU and without connecting of the external memory (see Figure 1 b).

To create such solution however, we need to analyze the constraints and requirements associated with the attempts to ensure reliable data transfer between FPGA and a computer.

High resources consumption of the TCP/IP based solution results mainly from two facts. The first one is a high complexity of the TCP/IP stack, which results from its versatility. The TCP/IP is prepared to work in the heterogeneous network, where communicating systems may be connected via many routers, and many network segments, possibly of different type. Additionally the TCP/IP assumes, that the network between the communicating systems may be insecure, so different mechanisms are implemented to prevent possible malicious attacks.[10–15]

In our data transmission subsystem, we may assume, that we use only the Ethernet network, and that the network connection is physically protected (which should be anyway assured to grant reliability of the measurement system). This assumptions allow us to use much simpler protocol. It also appears (which will be analyzed later), that we don't need routing, which leads to further simplification of the protocol, but requires that both communicating systems are located in the same network segment or segments connected via Ethernet switches.

Next problem related to high resources consumption - namely - necessity to provide a lot of memory for buffering of data, is associated with the acknowledge/retransmission mechanism used to ensure reliable transmission. Required amount of memory is directly associated with the transmission speed and with the latency of packet acknowledgement. If we denote the transmission speed as R_{transm}, and the maximum acknowledge latency as $t_{ack\ max}$, the required memory buffer capacitance M_{buf} may be calculated from the following formula:

$$M_{buf} = R_{transm}\, t_{ack\ max} \tag{1}$$

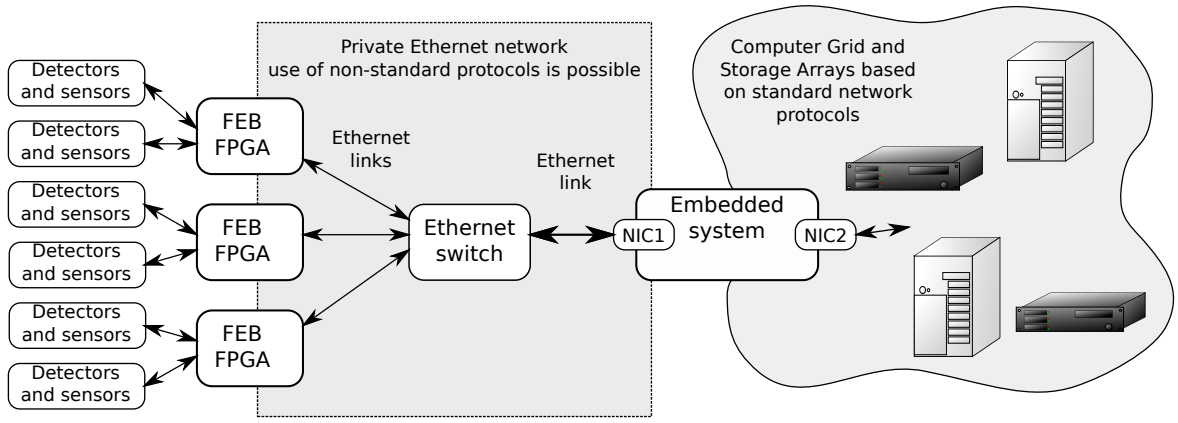

Figure 2. Architecture of the data transmission subsystem based on the FPGA connected to the Ethernet link and using the private, optimized Ethernet protocol, to transmit data to the embedded system. This embedded system receives data, preprocesses them, and sends them further for final processing or archiving, using standard network protocols.

Therefore if we want to transmit the data at certain speed, the only way to reduce the memory usage is to reduce the acknowledge latency.

Sample results of measurement of TCP/IP acknowledge latency are reported in [16]:

- 170 μs for Intel Core 2 T5500/1.66 GHz based system
- 240 μs for Pentium 4/2.8 GHz based system
- 520 μs for Ralink RT3350/320 MHz based system.

When routing is involved, the acknowledge latency increases above 1 ms. To ensure data buffering for the 1Gb/s transmission with the 2^{nd} of listed systems we need above 20 KiB of memory in case without routing, and above 120 KiB in case with routing. We must also consider, that the above results are just the average, not the "worst case" results.

The high latency of the TCP/IP acknowledge is the result of high complexity of the TCP/IP software stack implementation in the receiving computer (resulting from the already mentioned complexity of the protocol itself). If we want to use simpler algorithms, we can eliminate the TCP layer and use one of simpler IP protocols (e.g. UDP). However all IP protocols still require some overhead associated with routing of packets, which is simply not necessary in our system, as the routing leads to unacceptable increase of the acknowledge latency.

Therefore the optimal solution seems to be a maximally simple protocol, based on raw Ethernet frames. Such solutions have been already described,[17,18] but none of them is open (the sources are not available), and they do not seem to be focused on minimization of FPGA resources usage. Therefore the solution described in this paper was developed independently. The final architecture of the proposed data transmission subsystem is shown in Figure 2.

3. IMPLEMENTATION OF THE SYSTEM

The whole system consists of three parts - the network protocol, the FPGA IP core and the Linux kernel driver, which will be described in the following subsections. The previous version of the system, which uses more complex FPGA core, including the Ethernet MAC[19] is described in [16].

3.1 Network protocol

The protocol is based on raw Ethernet II frames with the ethertype set to `0xfade`. The ethertype `0xfade` was chosen arbitrarily, and is not officially registered, but as the proposed protocol is dedicated for non-routable private network, this solution seems to be acceptable.

The proposed protocol in its current version uses only four types of packets, shown in Table 1.

Table 1. Structure of the packets used by the transmission protocol. (SRC and TGT - MAC addresses of the transmitter and of the receiver)

packet	direction	structure
START packet	to FEB	SRC | TGT | 0xfade | 0x0001 | padding to 64 bytes
STOP packet	to FEB	SRC | TGT | 0xfade | 0x0005 | padding to 64 bytes
DATA packet	from FEB	SRC | TGT | 0xfade | 0xa5a5 | set number & packet number | delay | 1024 bytes of data
ACK packet	to FEB	SRC | TGT | 0xfade | 0x0003 | set number & packet number | padding to 64 bytes

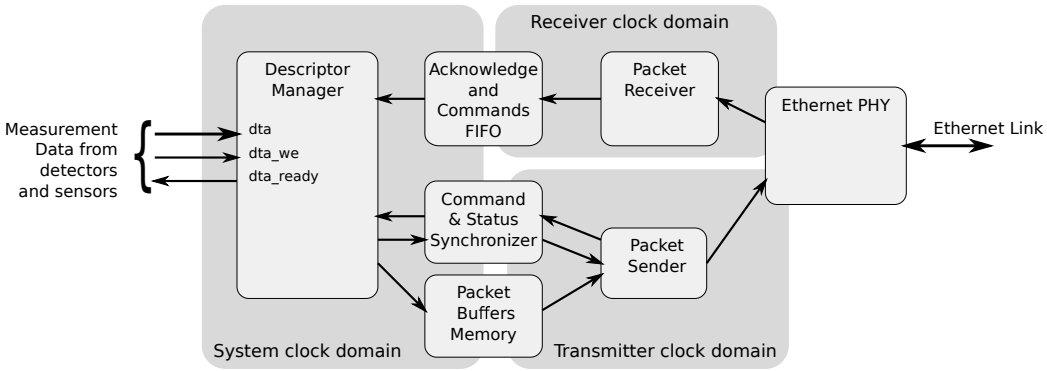

Figure 3. Blok diagram of the FPGA IP core used in the system.

The START packet is sent from the receiving computer, to initialize data transfer. After reception of this packet, the FEB with the MAC address corresponding to the TGT field in the packet, starts to send the data stream. The data stream is logically divided into *sets* (each set corresponds to the amount of data which can be buffered in the internal memory of FPGA). The *set* is further divided into *data packets* which are transmitted in the DATA packets. To simplify management of data both on the FPGA side and on the PC side, the *data packets* have length of 1024 bytes. In the version of the system described in this paper the *set* has length of 32 KiB and is divided into 32 *data packets*. Successful reception of the DATA packet is confirmed by the receiving computer with the ACK packet. The receiving computer may also request that the FEB should stop sending data, by transmitting the STOP packet with TGT field set to the MAC address of the appropriate FEB.

The protocol may be further extended to provide a method to send e.g. configuration settings to the FEB. Such extension is relatively simple and does not increase significantly the FPGA IP core.

3.2 Implementation of the FPGA IP core

The block diagram of the FPGA IP core is shown in Figure 3. The data acquired by the FEB are delivered to the synchronous *dta* bus, and when the system is ready to accept a new data word (which is signalled by the *dta_ready* signal), they are written with the *dta_we* strobe. This part of the IP core works in the *system* clock domain. The data are handled by the *Data Writer* state machine in the *Descriptor Manager*, which writes them to the dual port *Packet Buffers Memory*. This block processes *descriptors* associated with the *data packets*. Each *descriptor* stores the *set number* associated with the currently stored *data packet* and three flags describing its status: *valid* (V) - set when packet is filled with the new data, *sent* (S) - set, when the packet has been transmitted at least once and *confirmed* (C) - set when the packet has been confirmed by the receiver.

Another state machine in the *Descriptor Manager* - the *Data Reader* - cyclically browses this memory, and requests sending of the new packets and periodical retransmissions of the unconfirmed packets. Transmission of the packets is

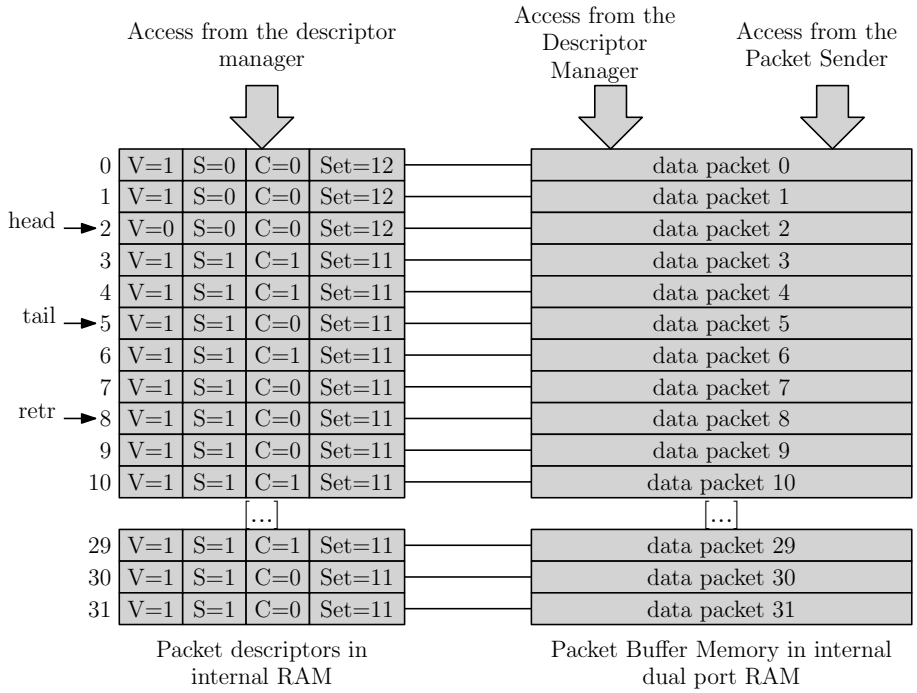

Figure 4. Sample content of the Descriptor Memory during the transmission. Packet buffers 0 and 1 are already filled with the data from the 12^{th} set, but have not been transmitted yet. Packet buffer 2 is still being filled with the new data from set 12 (and therefore its V flag is 0). Packet buffers 3 to 31 are filled with the data from the previous set 11. Packet buffers 3 and 4 are already transmitted and confirmed. Packet buffer 5 is transmitted but not confirmed. Therefore the *tail* pointer is set to 5. After this packet is confirmed, the *tail* pointer should move to position 7 (as packet 6 is already confirmed). The *retr* pointer circulates between the *tail* and *retr* positions, pointing to unconfirmed (C=0) packets which should be retransmitted.

handled by the *Packet Sender* state machine, which works in the *transmitter clock domain*. Signals transmitted between the *system* and *transmitter* clock domains are synchronized by dedicated synchronizers, and by a dual port memory. To avoid network congestion, a special delay is introduced between transmitted packets, and this delay is adaptively changed, as described in Section 3.2.2.

The confirmation and command packets (START, STOP, ACK) are received and parsed by the *Packet Receiver* state machine and the extracted information is written into the *Acknowledge and commands FIFO*, which assures synchronization between the *receiver* clock domain (in which the *Packet Receiver* works), and the *system* clock domain.

3.2.1 Acknowledge and retransmission system

The *Descriptor Manager* services the *Packet Buffer Memory* as a circular buffer. The *head* pointer points to the buffer which is filled with the data currently received from detectors and sensors. The *tail* pointer points to the last buffer, which contains not yet confirmed data. Initially, the *head* and the *tail* pointers are set to 0.

When the packet buffer pointed by *head* is filled completely, its V flag is set to 1 (the packet is marked as ready for transmission), and the state machines tries to move the *head* pointer to the next position. If this next position is pointed by the *tail* pointer, it means that there are no free buffers, and reception of the new measurement data must be suspended, until the next buffer is freed. In this situation the *dta_ready* signal is set to 0, and the *Data Writer* machine keeps waiting until the *tail* pointer is moved.

If the *head* may be moved to the next position, the descriptor pointed by it is set to the initial state (V=0, S=0, C=0), its *set* number is increased by 1, and the *dta_ready* signal is set to 1.

The buffer is freed, after its data are transmitted and confirmed. This task is performed by the *Data Sender* state machine, which cyclically browses the section of the *Descriptor Memory* located between positions pointed by the *head*,

and *tail* pointers (associated with the group of packet buffers filled with the data, and at least partially not confirmed). The third *retr* pointer is used to point to the buffer, which may be transmitted or retransmitted.

When the *Data Sender* finds a packet in the state V=1, C=0 (the S flag may be in any state, it is used only by the network congestion avoidance algorithm, described in Section 3.2.2), it requests that the *Packet Sender* transmits it. After the packets is sent, the S flag in its *descriptor* is set to 1. The same state machine also browses the *Acknowledge and Commands FIFO*. And if it finds the confirmation of any not confirmed packet (C=0), it sets its C flag to 1. If the packet pointed by the *tail* pointer is confirmed, it may be freed. The *tail* pointer is then moved to the next position. If the packet at this new position is also already confirmed, the process repeats, until *tail* points to the packet which is either not confirmed, or is pointed by the *head* pointer (the latter means, that all packets filled with the data are transmitted and confirmed). The described mechanism cyclically retransmits packets which have not been confirmed yet. In fact it is equivalent to the sliding-window mechanism used in TCP/IP with constant length of the window, equal to 32 packets. To avoid network and receiver overload, the consecutive transmissions are separated with adaptively adjusted delay, as described in Section 3.2.2. When the above described process leads to the situation where *tail* pointer is equal to the *head* pointer (all filled packets are transmitted and confirmed), the transmission process is suspended. Example of the contents of the Descriptor Memory during operation of the described algorithm is shown in Figure 4.

3.2.2 Avoiding of network congestion

Even though the system is supposed to work with the full duplex Ethernet network, it is possible that some packets may be dropped by the switch when multiple FEBs connected to the same switch are sending packets at the maximum rate. It is also possible that the receiving computer may not be able to handle and confirm all incoming packets, and may drop some of them. Both described problems lead to the situation when some packets are unconfirmed and should be retransmitted. However immediate retransmission of such not confirmed packet may further overload the network or the receiving computer and further increase the risk of packet loss which results in network congestion and reduction of the throughput.

To minimize the described problem, the system monitors the ratio between the numbers of sent packets ($C_{pkt\ sent}$) and number of retransmitted packets ($C_{pkt\ rsnt}$) (they are distinguished by the value of the S flag, when the packets is sent or resent). It is assumed, that if the packet is resent, it means, that either this packet, or its acknowledge packet got lost. There are two thresholds defined (T_{high} and T_{low}) setting the range of acceptable lost packet ratio. If the measured ratio is below the lower threshold T_{low}, the delay between packets is decreased. If the measured ratio is above the upper threshold T_{high}, the delay between packets is increased:

$$\text{if } \frac{C_{pkt\ rsnt}}{C_{pkt\ sent}} > T_{high} \text{ then } t_{del} := t_{del} \cdot \alpha_{incr} \tag{2}$$

$$\text{if } \frac{C_{pkt\ rsnt}}{C_{pkt\ sent}} < T_{low} \text{ then } t_{del} := t_{del} \cdot \alpha_{decr} \tag{3}$$

The coefficients α_{incr} and α_{decr} fulfill the conditions: $\alpha_{decr} < 1.0$ and $\alpha_{incr} > 1.0$.

This algorithm ensures, that the lost packet ratio remains in defined acceptable range. It is important that such algorithm may be fully implemented in the FPGA, and does not require any additional communication between different FEBs connected to the same receiving computer.

3.3 Implementation of the Linux kernel driver

To minimize latency of the packet acknowledgement, handling of the received packets has been implemented in the Linux kernel module*. However, as more complex operation associated with the data processing should be rather implemented in the user space, mechanisms allowing efficient data exchange with the user space applications have been also provided.

The driver may service multiple FEBs connected via network switch to the same network interface card (NIC), or to different NICs. The maximum number of serviced FEBs is defined, when the module is loaded, using the *max_slaves* parameter.

*In initial studies it was attempted to receive and confirm packet in the user space, using the *pcap* library, but the acknowledge time was very high, and therefore the link was not used optimally

Table 2. The *ioctl* commands implemented in the kernel module to support communication with the user space application.

IOCTL code	Description of the commands
L3_V1_IOC_SETWAKEUP	Sets the number of new data which must be present in the circular buffer, before the user space application will get woken up.
L3_V1_IOC_GETBUFLEN	Returns the length of the circular buffer associated with the particular FEB.
L3_V1_IOC_READPTRS	Returns the number of the available data bytes and the positions of the *head* and *tail* pointers in the circular buffer associated with the particular FEB. Provides necessary synchronization when accessing the pointers.
L3_V1_IOC_WRITEPTRS	Should be called with the number of bytes processed by the application. Provides necessary synchronization and updates the *tail* pointer in the circular buffer associated with the particular FEB.
L3_V1_IOC_STARTMAC	Associates the FEB identified by the given MAC, connected to the given network interface, with the particular character device, and starts the transmission
L3_V1_IOC_STOPMAC	Stops the transmission from the FEB associated with the particular character device.

For each serviced FEB a separate character device (*/dev/l3_fpga0*, */dev/l3_fpga1* and so on) is created. Each of those devices is associated with a separate circular buffer, which may be mapped using the *mmap* function into the application memory. Such approach minimizes overhead associated with accessing the data from the user space and is more efficient, than using the standard socket interface. However to provide proper synchronization when buffer pointers are accessed, and to allow suspending execution of the application when new data are not available yet, some functionalities had to be implemented using the *ioctl* function. The list of implemented *ioctl* commands with short descriptions is provided in Table 2.

3.4 Implementation of the circular buffer in the kernel module

Each FEB is associated with it's own circular buffer, managed with its own *head* and *tail* pointers. The data stream is transmitted as the continuous sequence of *sets*, and therefore the buffer management may be significantly simplified by using a buffer with length equal to a multiple of *set* length. Such approach warrants, that each *set* always occupies a contiguous area in the circular buffer. As it can be deduced from Figure 4, the packets being sent or resent at any particular moment, belong at most to two consecutive *data sets*. Therefore the kernel module may remember the positions of those *sets* in the circular buffer, minimizing the time needed to copy data when the DATA packet associated with one of currently handled *sets* is received.

The example explaining handling of the incoming packets by the kernel module is shown in Figure 5.

When a new packet with `0xfade` Ethertype is received, the driver checks if this is a command (START or STOP) packet and if yes, appropriately starts or stops the transmission. Otherwise, it checks if this is a valid data packet. If this is a valid and already confirmed data packet, (it means that it must have been retransmitted), the acknowledgement is sent immediately, to handle a situation when previous acknowledgement got lost. If this is a valid data packet which has not been confirmed yet, but it has a reasonable set number, the driver tries to copy the data from the packet to the appropriate part of the circular buffer corresponding to the particular FEB. After copying of data is completed successfully, the packet is marked as confirmed, and the appropriate ACK packet is sent.

Reception of packets with unexpected set numbers is considered as a symptom of serious error either in the software or in the hardware part of the system. Therefore in such situation module signals an error to the application and sends the STOP packet to the FEB, requesting to stop further transmission.

3.5 Minimization of the acknowledge latency

To minimize the time between reception of the packet and sending of the acknowledge packet, handling of the received packets is implemented in the *protocol handler*, installed with the *dev_add_pack* function [20, chap. 13]. Such solution allows to fully utilize functionality of the Network Interface Card device drivers and avoid delays associated with processing of packets in the standard network stack. The *protocol handler* hijacks all packets with the ethertype *0xfade* as soon as they are passed to the *netif_receive_skb* [20, chap. 10].

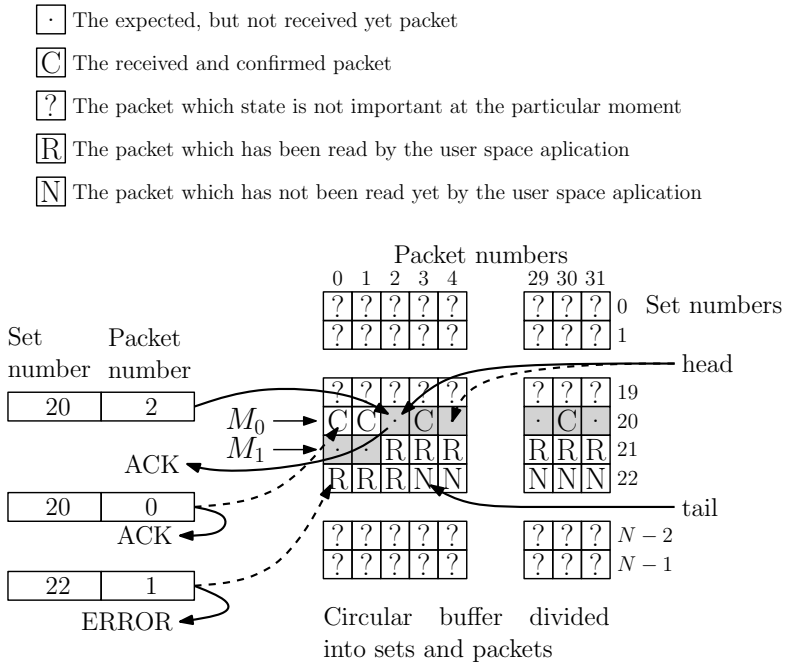

Figure 5. Handling of the received packets in the kernel module. The grayed area is the current position of the transmission window (starting at (20,2, and ending at (21,1)). The M_0 and M_1 pointers reference the beginnings of the currently received *sets* in the circular buffer.
When the packet (20,2) is received, its data are first copied into the circular buffer, and afterwards the ACK packet is sent. Reception of this packet will also move the *head* pointer to the (20,4) position, and shift the transmission window to (20,4)-(21,3).
When the packet (20,0) is received, the ACK packet is sent immediately (this packet is already received and confirmed, probably it was retransmitted before the FPGA received ACK, or the previous ACK got lost).
Reception of the (22,1) packet is a symptom of erroneous operation of the system. Only packets from sets 20 and 21 are expected. In fact also delayed packets from the set 19 could be tolerated, but packet from the "future" set 22 is obviously not expected.

4. RESULTS AND DISCUSSION

The system was tested using the Dell Vostro 3750 (Intel Core i7-2630QM CPU, 2.0 GHz clock) computer running the Debian/testing Linux OS (simulating the embedded system). Use of computer with 4-core CPU (with hyperthreading capable cores) allowed to confirm, that the code works reliably in multiprocessor environment. The FPGA based FEBs were simulated with three evaluation boards:

- SP601 evaluation board[2] equipped with 10M/100M/1G Ethernet PHY
- Atlys board[3] equipped with 10M/100M/1G Ethernet PHY
- Spartan-3E Starter Kit[1] (further denoted as S3ESK) equipped with 10M/100M Ethernet PHY

The current version of the system, based on the simplified Ethernet controller driving directly the external Ethernet PHY, was compared to the previous version, described in [16].

4.1 FPGA synthesis results

The IP core was successfully compiled for all FPGA platforms used for tests. The resources consumption for different platforms is shown in Table 3. As can be seen, in all tested FPGAs, synthesis of our IP core leaves significant amount of resources for another, user defined functionalities.

It is also visible, that replacement of the the OpenCores MAC core[19] with the simplified core, communicating directly with the Ethernet PHY, leads to decrease of resources consumption (by ca. 20% in case of boards with Spartan 6 chips, and by ca 7% in case of the Spartan 3E chip).

Table 3. Results of compilation of the IP core for different platforms

Board	FPGA chip	with OpenCores MAC[16,19]		without MAC	
		Slice usage	RAM usage	Slice usage	RAM usage
SP601	xc6slx16	501 out of 2278 (21%)	RAMB16BWER: 20 out of 32 (62%)	420 out of 2278 (18%)	RAMB16BWER: 18 out of 32 (56%)
Atlys	xc6slx45	542 out of 6822 (7%)	RAMB16BWER: 20 out of 116 (17%)	426 out of 6822 (6%)	RAMB16BWER: 18 out of 116 (15%)
S3ESK	xc3s500e	1355 out of 4656 (29%)	RAMB16: 20 out of 20 (100%)	1217 out of 4656 (26%)	RAMB16: 19 out of 20 (95%)

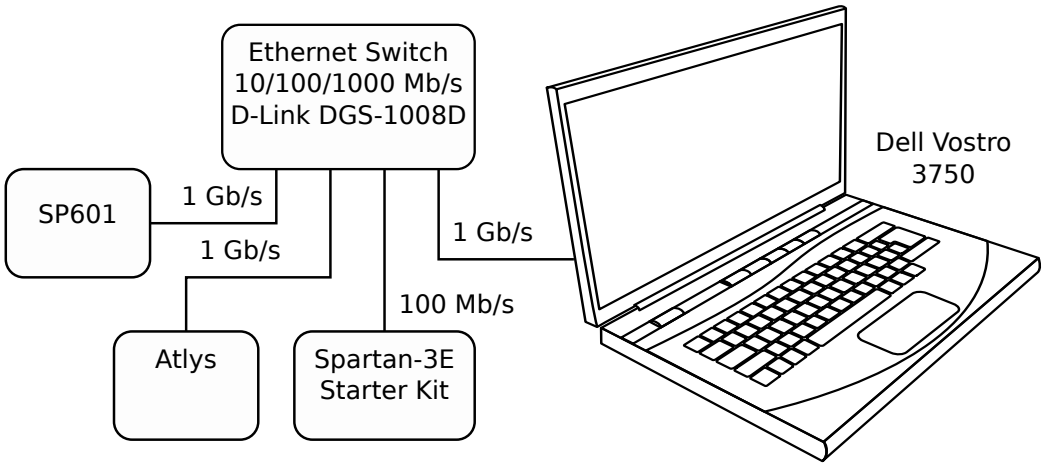

Figure 6. Test setup for throughput measurement.

4.2 Transmission tests results

The transmission tests were performed in the test setup shown in Figure 6. All tested boards were connected to the 10/100/1000 Mb/s Ethernet switch D-Link DGS-1008D,[21] and each board was assigned a unique MAC address. The data transmitted by simulated FEB were received by the user space application, which verified correctness of received data and confirmed their processing, freeing the buffer.

The tests included:

- measurement of the throughput
- capture of packets sent and received by the computer, and analysis of the acknowledge latency and "delay" value reported by the emulated FEBs in the data packets (using the wireshark[22] tool)

The tests were performed with the 32 KiB of internal RAM in each FPGA used for packet buffers (32 packets in a single set).

Results presented in Table 4 were obtained for parameters of the network congestion avoidance (NCA) algorithm set to $T_{high} = 1/8$, $T_{low} = 1/32$, $\alpha_{incr} = 1.25$, $\alpha_{decr} = 0.75$. The lost packet ratio was evaluated after transmission of every 10000 packets, and the delay was appropriately adjusted. The transmission speed was measured during 5 minutes to minimize influence of initial adjustment of the inter-packet delay.

For each combination of active boards 9 measurements were performed, and both average and standard deviation of transmission rate was calculated.

Table 4. Results of measurement of efficient transmission speed

Active boards	Measured efficient transmission speed [Mb/s]							
	Version with MAC (old)[16,19]				Version without MAC (new)			
	SP601 (1 Gb/s)	Atlys (1 Gb/s)	S3ESK (100 Mb/s)	Total	SP601 (1 Gb/s)	Atlys (1 Gb/s)	S3ESK (100 Mb/s)	Total
All boards active	451.2 σ=14.4	324.5 σ=17.7	91.3 σ=1.41	**867.0 σ=6.78**	385.9 σ=37.4	416.0 σ=37.6	86.7 σ=0.181	**888.6 σ=1.50**
SP601 and Atlys active	481.2 σ=25.2	371.2 σ=32.1	–	**852.4 σ=9.79**	446.4 σ=29.9	461.1 σ=25.9	–	**907.5 σ=4.97**
SP601 and SK3E	671.9 σ=13.0	–	91.9 σ=0.596	**763.8 σ=13.6**	711.3 σ=4.68	–	93.4 σ=0.296	**804.7 σ=4.50**
Atlys and SK3E	–	718,4 σ=4.41	93.9 σ=0.172	**812.4 σ=4.57**	–	715.2 σ=1.85	93.0 σ=0.141	**808.2 σ=1.73**
SP601 alone	891.9 σ=1.78	–	–	**891.9 σ=1.78**	921.2 σ=0.264	–	–	**921.2 σ=0.264**
Atlys alone	–	883.5 σ=1.18	–	**883.5 σ=1.18**	–	920.7 σ=0.675	–	**920.7 σ=0.675**
SK3ESK alone	–	–	95.0 σ=0.0001	**95.0 σ=0.0001**	–	–	94.5 σ=0.0001	**94.5 σ=0.0001**

Thanks to the careful adjustment of the receiving system parameters, and running the user space application with the real-time scheduling policy at high priority, it was possible to obtain much higher throughputs than those reported in [16].

The results show, that for transmission from a single board (rows 5, 6 and 7 in Table 4), the system achieves throughput near to the maximum throughput of the Ethernet PHY available in the particular board. Comparison of results for the new version of the system (with simplified Ethernet controller) and for the old version of the system (with MAC controller) shows, that removal of the MAC core and direct control of the PHY chip with a state machine resulted in increase of achievable transmission speed in case of boards with 1Gb/s PHY (rows 5 and 6 in Table 4), which can be explained by simplified flow of data in the FPGA without MAC. What is interesting, the transmission speed from the SK3E board, which is equipped with the 100Mb/s PHY has slightly decreased (row 7 in Table 4). The change is very small, but clearly visible. This effect is not explained yet and requires further analysis. Similar affect occurs also in case of simultaneous transmission from the Atlys board and the SK3E board (row 4 in Table 4).

All results show, that the link throughput is utilized in more than 80%, which proves, that the proposed system may be effectively used to transmit the data from FPGA to an embedded system. It can be also seen that the fluctuations of the total bandwidth utilization is much smaller in case of the system with simplified Ethernet controller (compare values of σ in the "Total" column in Table 4).

Unfortunately it appears that the proposed algorithm for avoidance of network congestion, does not always ensure equal distribution of the available bandwidth between different boards (see rows 4 and 2 in Table 4, the latter only for the old version of the system), even though it well stabilizes the total throughput. It suggests, that further research on optimum method for avoiding of network congestion is needed.

The mean acknowledge latency measured with the wireshark tool, was equal to 3 μs, which is significantly lower then latencies measured for the same computer using the TCP/IP protocol (see Section 2).

5. CONCLUSIONS

The presented system consisting of the simple network protocol, the optimized Linux kernel module, and the specialized FPGA IP core, may be used to provide reliable transmission of measurement data via Ethernet link from the FPGA based

Front End Board to an embedded system, which can preprocess the data, and send them further using standard network protocol.

Thanks to simple implementation of the FPGA IP core, and reduction of the memory requirements (due to minimization of packet acknowledge latency in the kernel module), it is possible to transmit data even from small FPGA chips, without necessity to use an external memory to buffer the transmitted, and not confirmed yet data.

The received data are available for the user space application through the memory mapped circular buffer, which significantly reduces overhead associated with copying of data from the kernel space to the user space.

The software components of the system are implemented in a SMP-safe way, which allows to use them in the multi-processor system, to increase the data processing speed.

System is capable to receive data from multiple FPGA based FEBs connected via network switch to the same Network Interface Card in the receiving system, but further research on network congestion avoidance algorithm is needed to improve distribution of bandwidth in such configuration.

Sources of the presented system are freely available under open source licenses on the OpenCores website.[23]

REFERENCES

[1] "Spartan-3E Starter Kit." http://www.xilinx.com/products/boards-and-kits/HW-SPAR3E-SK-US-G.htm (September 2012). [Online; accessed 15-September-2012].

[2] "Spartan-6 FPGA SP601 Evaluation Kit." http://www.xilinx.com/products/boards-and-kits/EK-S6-SP601-G.htm (September 2012). [Online; accessed 15-September-2012].

[3] "Atlys™ Spartan-6 FPGA Development Board." http://www.digilentinc.com/Products/Detail.cfm?NavPath=2,400,836&Prod=ATLYS (September 2012). [Online; accessed 15-September-2012].

[4] "IEEE Standard for Information Technology–Telecommunications and Information Exchange Between Systems–Local and Metropolitan Area Networks–Specific Requirements Part 3: Carrier Sense Multiple Access With Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications - Section One," *IEEE Std 802.3-2008 (Revision of IEEE Std 802.3-2005)* , c1 –597 (26 2008).

[5] "IEEE Standard for Information Technology–Telecommunications and Information Exchange Between Systems–Local and Metropolitan Area Networks–Specific Requirements Part 3: Carrier Sense Multiple Access With Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications - Section Two," *IEEE Std 802.3-2008 (Revision of IEEE Std 802.3-2005)* , 1 –790 (26 2008).

[6] "IEEE Standard for Information Technology–Telecommunications and Information Exchange Between Systems–Local and Metropolitan Area Networks–Specific Requirements Part 3: Carrier Sense Multiple Access With Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications - Section Three," *IEEE Std 802.3-2008 (Revision of IEEE Std 802.3-2005)* , 1 –315 (26 2008).

[7] "High Performance TCP/IP on Xilinx FPGA Devices Using the Treck Embedded TCP/IP Stack." http://www.xilinx.com/support/documentation/application_notes/xapp546.pdf (December 2004). [Online; accessed 15-September-2012].

[8] "LightWeight IP (lwIP) Application Examples." http://www.xilinx.com/support/documentation/application_notes/xapp1026.pdf (April 2011). [Online; accessed 15-September-2012].

[9] Uchida, T., "Hardware-based tcp processor for gigabit ethernet," *Nuclear Science, IEEE Transactions on* **55**(3), 1631–1637 (2008).

[10] Miller, I., "Protection Against a Variant of the Tiny Fragment Attack (RFC 1858)." RFC 3128 (Informational) (June 2001).

[11] Touch, J., "Defending TCP Against Spoofing Attacks." RFC 4953 (Informational) (July 2007).

[12] Eddy, W., "TCP SYN Flooding Attacks and Common Mitigations." RFC 4987 (Informational) (Aug. 2007).

[13] Gont, F., "ICMP Attacks against TCP." RFC 5927 (Informational) (July 2010).

[14] Ramaiah, A., Stewart, R., and Dalal, M., "Improving TCP's Robustness to Blind In-Window Attacks." RFC 5961 (Proposed Standard) (Aug. 2010).

[15] Gont, F. and Bellovin, S., "Defending against Sequence Number Attacks." RFC 6528 (Proposed Standard) (Feb. 2012).

[16] Zabolotny, W. M., "Efficient transmission of measurement data from FPGA to embedded system via Ethernet link." http://arxiv.org/abs/1208.4490 (Aug. 2012).

[17] Uchida, T., Fujii, H., Nagasaka, Y., and Tanaka, M., "New communication network protocol for a data acquisition system," *Nuclear Science, IEEE Transactions on* **53**(1), 286–292 (2006).

[18] Mindur, B. and Jachymczyk, L., "A general purpose ethernet based readout data acquisition system," in [*Nuclear Science Symposium and Medical Imaging Conference (NSS/MIC), 2011 IEEE*], 800–806 (2011).

[19] "10_100_1000 Mbps tri-mode ethernet MAC." http://opencores.org/project,ethernet_tri_mode (November 2011). [Online; accessed 15-September-2012].

[20] Benvenuti, C., [*Understanding Linux Network Internals*], O'Reilly Media (2006).

[21] "8 Port 10/100/1000Mbps Gigabit Switch DGS-1008D." http://www.dlink.com/uk/en/business-solutions/switching/unmanaged-switches/desktop/dgs-1008d-8-port-10-100-1000mbps-gigabit-switch (September 2012). [Online; accessed 15-September-2012].

[22] "Wireshark - the world's foremost network protocol analyzer." http://www.wireshark.org (August 2012). [Online; accessed 15-September-2012].

[23] "Fade - Light L3 Ethernet protocol for transmission of data from FPGA to embedded PC." http://opencores.org/project,fade_ether_protocol (May 2013). [Online; accessed 9-June-2013].