Ethernet-based slow control system for parallel configuration of FPGA-based front-end boards

Wojciech M. Zabołotny[1]

[1]Institute of Electronic Systems, Warsaw University of Technology

✉ wzab@ise.pw.edu.pl

Introduction

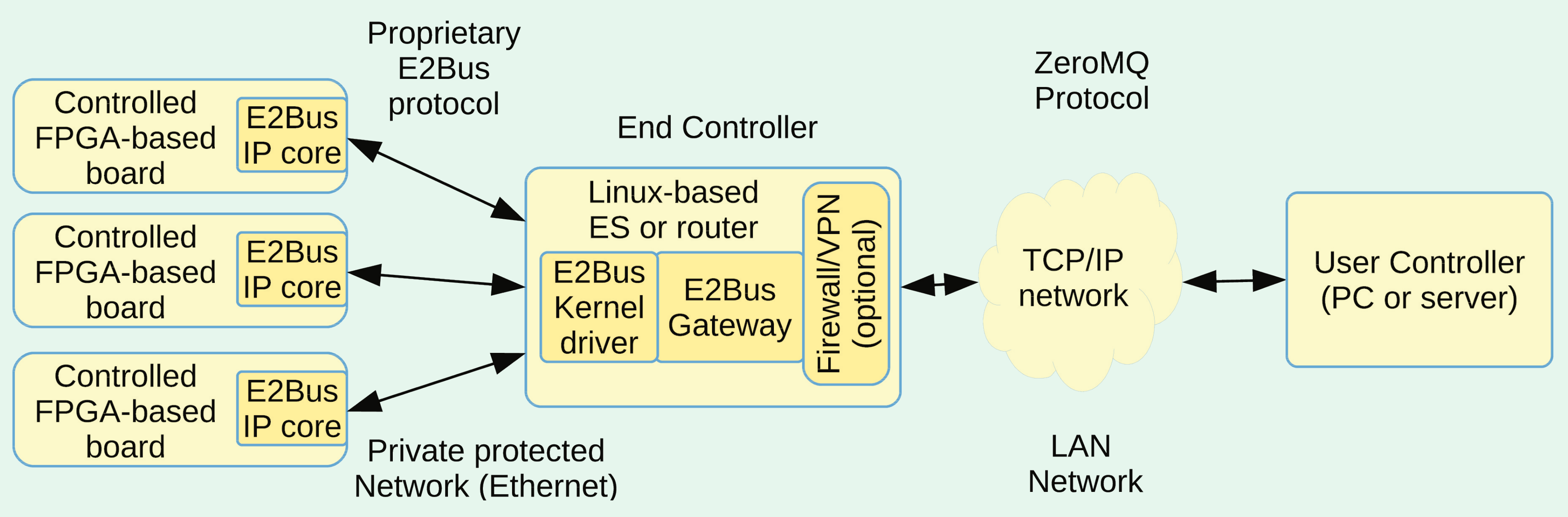
The idea to use the Ethernet interface to control the measurement systems is well established. The Ethernet network infrastructure is widely used, and due to mass production, it is reasonably priced. It enables creating the distributed systems using either copper or optical fiber connections.
Currently, the IPbus [1,2] is the Open Source standard for such control. It is mature, well tested and used in many applications. However, it seems that some assumptions underlying IPbus may result in unnecessarily increased complexity and reduced efficiency.
This work presents **E2Bus** - an experimental alternative solution aimed at reduction of resource consumption and improvement of achievable performance.

Main IPbus features

For communication with hardware, IPbus uses the UDP protocol over IPv4 via 1Gb/s Ethernet link. Use of UDP allows handling of packets in the user space at the computer side. The IP addresses may be assigned automatically based on MAC addresses, using the RARP protocol. UDP packets are routable. However, due to possible losses of the packets, this feature is rather not used. The reliable operation of IPbus is assured by Erlang-implemented ControlHub, which also converts the UDP-based communication into the TCP-based communication. TCP enables remote access (including tunneling if needed). Communication with the user applications is provided by the uHAL library, that allows interfacing with C++ or Python code. Standard makefiles for IPbus software support Scientific Linux and Centos. Due compiler and library dependencies it may be difficult to compile it for another Linux distributions.
To increase performance, IPbus allows combining multiple commands in a packet. It also provides simple Read-Modify-Write operations.

Possible improvements in E2Bus

It is possible to further simplify the FPGA layer by usage of layer 2 Ethernet frames instead of UDP. We lose routability, but for security reasons, the communication between the FPGA and the first computer system should be limited to a single, physically protected layer 2 segment. Such frames may be processed in user space (e.g with the libpcap library), but to improve performance it is better to handle them in the Linux kernel. That's especially important for implementation of reliable communication - handling of acknowledgments and retransmission of not confirmed frames.
The similar approach was used in FADE-10G protocol for transmission of measurement data [3].
E2Bus commands sets and responses are generated and processed in the user space using a simple C-based application and further transmitted to or from the remote controller using the ZeroMQ over TCP protocol.
That minimalistic approach allows using a very simple Linux machine, even a Linux-based router as an E2Bus "End Controller" (EC) (an equivalent of a machine running ControlHub in IPbus).
The possible architecture of E2Bus-based control system is shown in the figure.


Proprietary E2Bus protocol
ZeroMQ Protocol
End Controller
Controlled FPGA-based board
E2Bus IP core
Controlled FPGA-based board
E2Bus IP core
Controlled FPGA-based board
E2Bus IP core
Linux-based ES or router
E2Bus Kernel driver
E2Bus Gateway
Firewall/VPN (optional)
TCP/IP network
User Controller (PC or server)
Private protected Network (Ethernet)
LAN Network

E2Bus also extends the concept of combining of commands. The idea (described in [4]) is to allow embedding of simple tests in the set of commands. New commands are "read and test" that allows checking a condition and interrupt further processing if it is not met, and "multiple read and test" that allows waiting (with timeout) until a certain condition is met.
The IPbus does not support interrupts. The controlling software must poll the hardware. E2Bus removes that limitation.

E2Bus network protocol

Communication between the E2Bus End Controller and controlled FPGA boards is performed in a private layer 2 network segment, using the Ethernet frames with protocol ID set to 0xe2b5.
It is assumed that this network will be used only for E2Bus traffic, but it is not a strict requirement. Other protocols may be used in parallel. However, such configuration may impair performance.
E2Bus protocol defines two formats of packets:

Downlink frame (to FPGA)

Ethernet header	Protocol header	RESP ACKs (optional)	Command set (optional)	CRC

Uplink packet (from FPGA)

Ethernet header	Protocol header	CMD ACKs (optional)	IRQ status (optional)	Response data (optional)	Filler (if needed)	CRC

The protocol header contains the 0xe2b5 word followed by the protocol version (now 0x0001).
Command sets and response data records are marked with 15-bit sequence numbers. In the downlink packets, the response acknowledgments are sent as 16-bit words with the 15th bit set, that allows to distinguish them from the command set, that starts with 0x5a followed by the length of the set, and the commands. The special commands, like RESET, OPEN and CLOSE are transmitted as 0x5b followed by the command code.
In the uplink packets, similarly, the command acknowledgments are transmitted as 16-bit words with the 15th bit set. The IRQ status is transmitted as a 16-bit word with 0x59 in the first byte and status of 8 IRQ lines in the second byte. It is only transmitted if any IRQ line is active. The response data starts with 0x5a. The following header contains the 15-bit number of response data set, lower 8-bits of the associated command set, and the length of the set. Certain commands like block read may generate long responses that must be transmitted in multiple packets. Therefore, the header also contains information if the packet contains the last response associated with the particular command set.
The uplink packets are sent if there is an active interrupt, if a new command set is received, or if there exists any unsent or unconfirmed response set. The delay between consecutive retransmissions and IRQ notifications is configurable.

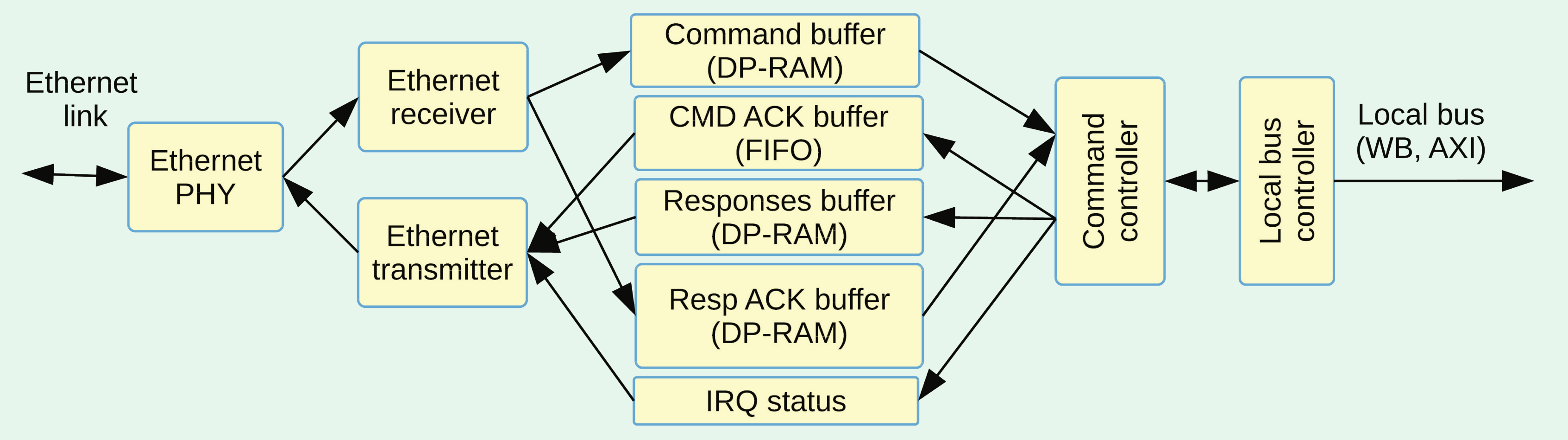
Implementation of End Controller

The End Controller (EC) is a Linux-controlled computer or embedded system. The requirements of E2Bus are so low that it is possible to implement the EC in a Linux-based router or access point, combining functionalities of the network device with EC.
The E2Bus kernel driver (e2bus.ko) is simple, so it is possible to implement control algorithms in a C application directly communicating with the driver. However, the standard solution is to run the E2Bus gateway (e2bus_gw) that provides communication with E2Bus via ZeroMQ protocol [5].

- It still allows implementing the User Controller (UC) on the same machine using the IPC transport.
- Usage of TCP transport enables remote connection of UC.
- It is possible to use either internal ZMQ encryption or VPN or tunneling to provide secure control in public network.
- For the execution of the commands, the User Controller connects to the End Controller using the ZMQ PAIR pattern.
- For notification about interrupts, the User Controller connects to the End Controller using the ZMQ Publish/Subscribe pattern.

Implementation of E2Bus IP core

The architecture of the E2Bus IP core responsible for the FPGA side of the E2Bus protocol is shown in the figure below.



Command controller

The important part of the solution is the command processor, that executes the whole sets of commands. It is a practical implementation of the concept described in [4]. During the execution of the command set, the controller generates the response. When the response is too long to fit in a single packet, the packet is transmitted, and filling of the next packet starts.
In the last response packet, at the end, there is a status of the operation. In case of error, there is also the address of the last executed command that produced that error.
After the error, next commands are not executed, and next command sets also are rejected with the error. The error condition is cleared only when a command set beginning with a special "ERROR-CLEAR" command is received. That ensures reliable operation in the mode with multiple packets.

The following operations are supported :

- READ: Single or multiple (up to 4096) reads are possible. The read address may be incremented, decremented or constant. Each read writes the read value to the response.
- WRITE: Single or multiple (up to 250) writes are possible. The write address may be incremented, decremented or constant. The whole command including data must fit in a single 1024-byte block. No response is produced.
- READ-MODIFY-WRITE: Possible operations include: Increment, Decrement, Add, Subtract, And, Or, Xor, Not. It is possible to write the original value to the response.
- READ-AND-TEST: Possible tests include: Signed/unsigned less/greater than, compare, and/or with mask and compare. If the condition is not met, the calculated value is written to the response, and error is generated.
- MULTIPLE-READ-AND-TESTS: The same tests as above are possible. The test may be repeated the programmed number of times with the programmed delay. If the test is finally passed, the number of repetitions left is written to the response. If not, the value calculated in the last repetition is written to the response, and error is generated.

Implementation of the E2Bus kernel driver

To minimize the acknowledgment and retransmission latency, the kernel driver installs its private protocol handler in the Linux kernel. Communication with the user space application (usually the e2bus_gw) is done via ioctl calls.
Communication is started with E2B_IOC_OPEN, which establishes the connection with the FPGA board basing on its MAC address. It also informs the E2Bus IP core (EBC) about the MAC of the End Controller and resets the EBC (without resetting of controlled peripherals).
When the board is connected, the user space application may submit the list of commands for execution. The list should not be longer than 1024 bytes. It may be submitted for synchronous execution (E2B_IOC_SEND_SYNC) or for asynchronous execution (E2B_IOC_SEND_ASYNC). The latter allows submission of multiple sets in a sequence for maximal performance. The application may wait until the command set is executed using E2B_IOC_RECEIVE ioctl. The standard poll function is also supported for single threaded e2bus gateways.
To minimize copying of data, the received response is written to the buffer provided by the user space application and mapped using the get_user_pages function after the appropriate E2B_IOC_SEND_xxxx call.
The ioctl E2B_IOC_WAIT_IRQ function allows waiting in a separate thread for interrupts.

Results

The first "proof of the concept" implementation of the proposed E2Bus system has been tested.
The E2Bus IP core has been implemented as ISE project for Spartan 6 (2691 LUTs, 9 BRAMs) and as Vivado project for Artix 7 (2829 LUTs, 6 BRAMs) FPGA. Versions for 1 Gb/s and 100 Mb/s have been prepared. The E2Bus kernel driver and E2Bus gateway have been compiled and tested on the Intel x86 platform and ARM platform. The sources were converted to Buildroot packages to allow easy testing on embedded platforms. The OpenWRT version is still under preparation.
The correct operation of the system with a simple set of WishBone slaves has been proven.
The library for automatic generation of command sets and interpretation of responses (an equivalent of IPbus uHAL library) is still in preparation.
E2Bus will be released as Open Source project. The sources will be available at [6].

Conclusions

The proposed E2Bus system may be used to build distributed Ethernet-based control systems for FPGA-based boards. Usage of layer 2 Ethernet frames enables simplification of the FPGA IP core and low-latency implementation of reliable transport in the Linux kernel with own packet handler. Implementation of the protocol in a kernel driver and minimal ZeroMQ server enables the use of even simple Linux-based routers as End Controller nodes in the system.

Parallel control of multiple boards is supported by the following features:
- It is possible to submit multiple packets with commands sets for execution. The next set is processed, while the results produced by the previous ones are transmitted.
- Simple handshake operations, like waiting for certain bits to be set or cleared are executed locally by the E2Bus IP core, without generating additional latency and network traffic. Error conditions stop the execution of the whole submitted set of commands. Detailed information about the error is included in response for recovery procedure.
- Interrupts support reduces the need for polling of the controlled hardware

The proposed solution is still not mature. However, the tests of the first implementation have proven the correctness of the concept. Further tests and cleanup of the code are required.

Acknowledgment

Work was supported by statutory funds of Institute of Electronic Systems.

References

[1] C.G.Larrea, K.Harder, et al. "IPbus: a flexible Ethernet-based control system for xTCA hardware", JINST 10(02)C02019, 2015, doi:10.1088/1748-0221/10/02/C02019
[2] R. Frazier, G. Iles et al. "Software and firmware for controlling {CMS} trigger and readout hardware via gigabit ethernet", Physics Procedia 37 (2012), pp. 1892 – 1899; doi:10.1016/j.phpro.2012.02.516
[3] W.Zabolotny, "Low latency protocol for transmission of measurement data from FPGA to Linux computer via 10 Gbps Ethernet link", JINST 10(07)T07005, 2015, doi:10.1088/1748-0221/10/07/T07005
[4] W.Zabolotny, "Improvement of FPGA control via high speed but high latency interfaces", Proc. SPIE, 9662, 96623G-96623G-8,2015,doi:10.1117/12.2205441
[5] "ZeroMQ Distributed Messaging", http://zeromq.org/
[6] "E2Bus - control of FPGA-based systems via Ethernet interface", https://github.com/wzab/e2bus