Introduction

The idea to use the Ethernet interface to control the measurement systems is well established. The Ethernet interface is embedded in almost every board, and due to mass production, it is reasonably priced. It enables creating the distributed systems using either copper or optical fiber connections.

Currently, the IP[2,1] is the OpenFlow protocol for such control, and it is widely used and in many applications. However, it seems that some assumptions underlying IP 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 IPbuses features

For communication with hardware, IPbus uses the UDP protocol over IPv4 via GiBus Ethernet link. Use of UDP allows handing of lossy link to user space eventing, which may be assembled statistically 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 as a statistical application-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 Ubuntu. Installation is relatively simple and can be compiled for another Linux distributions.

The OpenFlow protocol, 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 know that reliability, but due to security reasons, the connection 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 Bifcap library), and if it improves 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 IPv6 network implementations of measurement data [3]. E2Bus commands and settings are generated and processed in the user space using a simple script, that allows transferring of those commands and settings from the remote controller using the ZeroMQ socket/CPP protocol.

The most realistic approach uses 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 is as follows. An E2Bus control system is shown in the figure.

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 IPID set to Oxes262. It is assumed that this network will be used only for E2Bus, and no other protocols may be used in it. However, such configuration may impair performance. E2Bus protocol defines the structure of packets:

Dowlink frame (to FPGA)

<table>
<thead>
<tr>
<th>Ethernet header</th>
<th>Protocol header</th>
<th>RESP/ACKs (optional)</th>
<th>Command set (optional)</th>
<th>CRC</th>
</tr>
</thead>
</table>

Uplink packet (from FPGA)

<table>
<thead>
<tr>
<th>Ethernet header</th>
<th>CMD ACKs (optional)</th>
<th>IRQ status (optional)</th>
<th>Response data (optional)</th>
<th>Filter (if needed)</th>
<th>CRC</th>
</tr>
</thead>
</table>

The protocol header contains the 8xe262 byte followed by the protocol version (now 0x0001). Command sets and response data records are marked with 15-bit sequence numbers. In the downstream packet, the command is sent as 16-bit words with the 15th bit set. It allows to distinguish them from the command set, that starts with 0x0a followed by the length of the set, and the commands. The special commands, like RESET, OPEN and CLOSE are transmitted as 0x05 followed by the command code.

In the upstream packet, 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 OxFF in the first byte and status bits of 08 (if the command is accepted) or 00 (if the command is rejected). The FIFO response data starts with 0x0a. The following header contains the 15-bit number of response data set, lower 8-bits of the sequence number, and the command set (the lower 7 bits). The response packets are sent 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 upstream packets are sent if there is an active listener. If a new command set is sent, or if there exists any unsent or unconfirmed response packet. The delay set between consecutive transmissions and the same command set is at least 1 second.

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.

Implementation of E2Bus Network driver

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 responses. 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 case of complete command, the controller is in the error state. 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 are also rejected with the error. The error condition is cleared only when a command set begins 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 value to the response.
- WRITE: Single or multiple (up to 256) 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 of the FPGA memory.
- READ/MODIFY/WRITE: Possible operations include: Increment, Decrement, Add, Subtract, And, Or, XOR, Exclusive OR, Write the original value. For these operations, the READ-TEST AND-TEST: Possible operations include: Signed/unsigned less/greater than, compare, and/or with mask and compare. If the condition is met, the written value is calculated to the response.
- MULTIPLE-READ-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.go) is done via ioctl calls.

Communication is started with E2B_IOCTL_OPEN, which establishes the connection with the FPGA board based on its MAC address. It also informs the E2Bus IP core (EBC) about the MAC of the End Controller and resets the E2Bus protocol processing of the commands.

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 with E2B_IOCTL_SEND/SEND, or for asynchronous execution with E2B_IOCTL_SEND/SEND. The latter allows submission of multiple sets in a sequence for maximal performance. The application may wait for the completion set via E2B_IOCTL_RECV/RECV. The standard poll function is also supported for single thread e2bus gateways.

To minimize copy data of command sets, and when written to the user space, application map and mapping the get_user_pages function page(s) with the appropriate E2B_IOCTL_RECV/SEND nakne.

The ioctl.E2B_IOCTL_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 (2639 LUTs, 6 BRAMs) FPGA. Versions for 1 Gbit and 100 Mbit have been proposed for the FPGA kernel driver and E2Bus protocol processor. The FPGA system was tested and compiled for the Xilinx 6 platform and ARM platform. The sources were converted to Buildroot packages to allow easy testing on embedded systems. The Open Source version is still under preparation.

The correct operation of the system with a simple set of Wishbone slaves has been proven. The test is for automatic parameters setup with the programmed delay. (an equivalent of IPbus UHAL library) is still in preparation.

E2Bus will be released as Open Source project. The sources will be available at [8].

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 allows 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 a part of 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 executed as soon as the current one is finished.
- Simple handshake operations, like waiting for certain bits to be set or cleared are executed locally by the E2Bus IP core, without any additional latency on the network traffic. Error conditions stop the execution of the whole submitted command set. Detailed information about the error is included in response for recovery procedure.
- Interrupts support reduces the need of opening 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