Improvement of FPGA control via high speed but high latency interfaces

Wojciech M. Zabołotny[a]

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

ABSTRACT

In last years, the throughput of interfaces used in computer systems to control extension boards or external hardware has increased significantly. Unfortunately, those interfaces have also significant round-trip latency. This fact seriously impairs the efficiency of those control algorithms, which require a tight handshake. In such algorithms, the communication consists of a sequence of write and read operations, where read result (the handshake status) must be checked before the next write command is issued. This problem can be solved by the implementation of an intelligent controller in the controlled hardware. This controller should execute high-level commands locally performing all necessary handshake operations. Unfortunately, such a complex and highly specialized controller would consume a significant amount of FPGA resources. This paper presents an alternative approach which uses a highly simplified versatile controller implemented in FPGA. This simple controller may improve the efficiency of certain, relatively broad class of control algorithms. The proposed controller accepts a set of simple commands, which describe the write operations, read operations, and simple test operations. The control algorithm is described as a sequence of those operations. If the controlled hardware works correctly, all tests are passed, and the controller only notifies the host about successful completion. In case if certain handshake test fails, the host is notified about the position of the failed test and type of failure. That allows the controlling software to investigate and cure the problem. The controller may be also used in a standard mode, where status or result of each command is returned immediately and may be checked before the next command is issued.

The paper also proposes a simple method for writing of software, which uses the new controller. This method allows to implement the control procedures that are very similar to those using traditional controllers. That minimizes the effort needed to use the new controller and reduces a risk of errors.

Keywords: FPGA, Control interfaces, Embedded systems, Measurement systems, Communication interfaces, Latency

1. INTRODUCTION

In modern measurement or control systems, it is usually necessary to control the external hardware from the computer system. It may be an extension board connected to the embedded system via a peripheral I/O interface or the complex, autonomous system controlled via long distance link (e.g. the Ethernet connection). Contemporary communication interfaces offer very high throughput, even up to a few Gb/s. It is a significant increase of bandwidth comparing to old interfaces, like VME which offered throughput up to 320 Mb/s. Unfortunately it appears, that this increase of throughput does not result in a proportional reduction of time of execution of typical control algorithms. Debugging of the problem shows, that even though those interfaces offer very fast block read or block write operations, their performance is significantly reduced if the user needs to perform single read and write operations. Further reduction of speed occurs, when the user needs to modify the control flow, basing on the results of a read operation. Unfortunately, this is a typical situation in most control algorithms which use handshake procedures to make sure, that the previous operation is successfully completed, and the hardware is ready for the next one. Those negative effects are tightly associated with the way, how the modern high-speed interfaces work. Therefore their elimination or at least mitigation is not trivial.

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

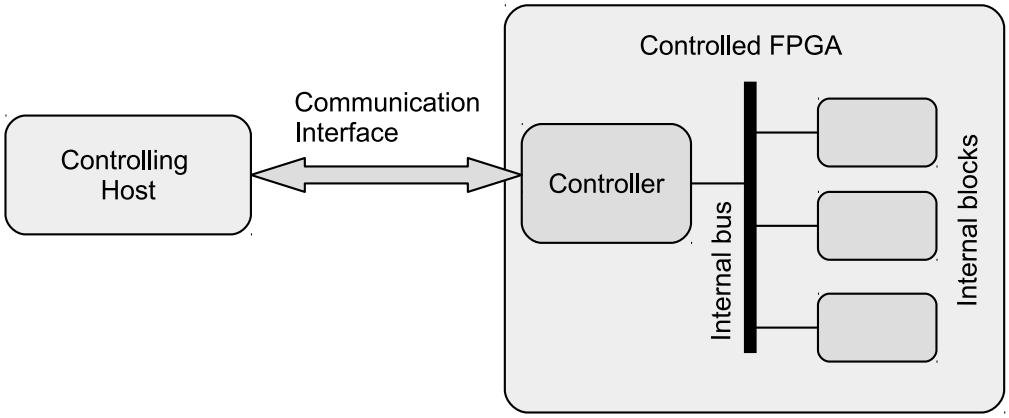


Figure 1: Typical architecture of the electronic system with FPGA controlled by a computer system.

2. ANALYSIS OF THE PROBLEM

For years, engineers were used to having random access to I/O registers. It significantly simplifies the implementation of hardware control algorithms. Let us analyze a typical sequence of operation found in a control algorithm:

- Write 0xa0 to register 0x10.
- Wait until bit 7 in register 0x11 is set. If it is not set after 10 μs, raise an exception.
- Write 0x00 to register 0x10.

If those registers are connected directly to the CPU bus, those operations may be performed at the full speed of the bus.

Now let us consider the same operation performed in the FPGA connected via such modern communication interface, as shown in Figure 1.

Even if the system uses a high-speed interface, the effective speed (measured as the number of operations performed in a unit of time) will be significantly reduced. The reason is that most modern high-speed interfaces are packet oriented. The overhead associated with the preamble, header and tail of the packet is not so significant when we send a packet containing multiple operations, as in a case of block read or block write operations. In case of a single read, like used in the above code snippet, this overhead may be much more significant. Additionally, the controller in the FPGA must wait with the execution of the command until the whole packet is received, and its checksum is verified. In case of read operation, sending of the result of even a single read operation also requires transmission of the whole packet. So when we use modern communication interfaces to implement control of our system, we can expect following properties:

- Block operations are much more efficient than single operations.
- Even though the throughput is high, the latency of the read operation may be significant.
- If we expect the hardware to acknowledge operations (e.g. to detect internal bus errors), the acknowledgment latency will be high.

Particularly the latency of the read operations may be significant. In case of the PCIe interface, it may be even up to 300 ns.[1,2] It means, that when performing multiple single read operations, the performance of the bus may be much worse, than in case of the old VME interface! The similar problem occurs in control systems based on the Ethernet interface. They are popular last time thanks to low cost and wide availability of the Ethernet hardware. An example of such solution may be the IPbus[3] developed at CERN. Unfortunately, the Ethernet-based solutions also suffer from the high round-trip latency.

Let us assume that a single read operation requires a sending of one command packet from the controlling host to the FPGA, and one response packet from the FPGA to the controlling host. With the minimal length of packets of 72 bytes (including preamble), at the bit rate of 1 Gbps it means, that the minimum round-trip latency is equal to:

$$2 \text{ packets} \cdot 72 \text{ bytes/packet} \cdot 8 \text{ bits/byte} \cdot 1 \text{ ns/bit} \; = \; 1152 \text{ ns}$$

That is much higher latency than in the case of both VME and PCIe. Another high-speed interface is the USB - often used for development and debugging of FPGA-based systems. Unfortunately, it also has the high read latency due to its packet-based transmission (in the version 2.0 and below additionally increased by the polling interval).[4,5]

3. POSSIBLE SOLUTIONS

The ultimate solution to the problem would be to implement an intelligent controller in the FPGA, that accepts high-level commands with associated data sets, performs the whole control algorithm with all necessary handshake at the full internal bus speed, and finally sends the results back to the controlling host. Unfortunately, such a solution is complex and resource hungry, especially if we would like to develop a versatile solution, able to perform different control algorithms. On the other hand, the dedicated controller optimized for particular system and control algorithm may be difficult to maintain. Each modification of the controlled hardware or control algorithm could require a redesign of such a controller. An attempt to provide a versatile, intelligent controller with reasonable complexity is described in.[6] Another approach may be to find a minimal controller still able to eliminate round-trip latency problems. To find such a solution let us review optimizations that have been proposed up to now.

3.1 FPGA side implementation of read-modify-write operations

The operations consisting of subsequent: read, modifications of the received value and write of the resulting value are very time inefficient when performed via high-latency interface. Therefore, servicing of such commands in the FPGA controller is an obvious optimization. Such solution is well known and used in the PCIe interface (as Atomic Operations limited to "Fetch and Add", "Swap" and "Compare and Swap"),[7] in the RapidIO interface (as Atomic "Increment", "Decrement", "Set", "Clear", "Swap", "Test and Swap" and "Compare and Swap"),[8] and in the IPbus protocol (as "Read-Modify-Write bits" and "Read-Modify-Write sum").[3]

3.2 Write caching

Another trivial optimization is caching of write operations. The write operations may be not executed immediately but accumulated and then sent in a single packet. Also, the transmission of their result from the FPGA to the controlling host may be delayed until an error occurs or until the first read operation is required. This approach has been used by the author when optimizing the JTAG interface controlled via the fiber-connected VME interface.[9] The similar approach is also used in the PCIe[7] in posted write operations. Generally, this optimization may help in case of algorithms, which consist mainly of write operations with rare read-based handshake operations. A practical example may be a configuration of FPGAs.

3.3 Write and read caching

In a broader class of control algorithms both read and write operations may be used, but only after certain read operations the handshake results are evaluated basing on results of previous reads. In case of such algorithm, the controlling host may send a longer sequence consisting of both read and write operations. The FPGA controller buffers results of read operations until the first error occurs or until it is necessary to check returned results. Therefore, the controlling host may speed up the control process using block transfers and wait only if handshake operation is required. In fact, this approach requires only implementation of the special *sync* command, which informs the FPGA controller, that the status of previous write commands and results of previous read operations must be transmitted to the host.

An example of such approach may be the implementation of the "Rlink" protocol implemented by W.F.J.Mueller in his "w11" project.[10] It allows to prepare lists of commands (both read and write) to be executed. Execution of the list may be aborted in case of an error. Whether the list is successfully completed or not, the results of its execution are sent back to the controlling host.

3.4 Minimal controller supporting test commands

In the typical control algorithm, results of read commands are used to verify the correctness of the hardware operation and/or to adjust timing. Considering that, the concept described in the previous subsection may be easily extended with basic test commands needed to perform the simple handshake. The result is a simple controller, able to perform autonomously fairly complex control algorithms almost at the local bus speed. Its performance is only minimally affected by the round-trip latency of communication interface.

4. IMPLEMENTATION OF THE PROPOSED FPGA CONTROLLER

The FPGA controller reads commands and data from the input buffer. That buffer stores the packet delivered from the controlling host by the communication interface. The results of read operations, possible error status generated by a write operation, and statuses of test operations are stored in the output buffer. The output buffer is used to build the response packet to be returned to the controlled host. The response packet is sent whenever the output buffer is full, or when an error occurred during read, write or test operation. To keep the FPGA controller as simple as possible, the minimal subset of operations should be implemented.

4.1 Write command (single or block)

This command accepts the following arguments:

Start address - Address of the first word to be written

Length of the block - Number of words to be written

Address modification - This argument defines whether, after each write, the address should be incremented, decremented or kept unchanged (for FIFO access)

Data words - Vector of data words with appropriate length

In case of the local bus error, the write command generates the exception with information about the address that couldn't be written and data word that was attempted to write.

4.2 Read command (single or block)

This command accepts the following arguments:

Start address - Address of the first word to be written

Length of the block - Number of words to be read

Address modification - This argument defines whether, after each write, the address should be incremented, decremented or kept unchanged (for FIFO access)

This command writes to the output buffer the vector of data words read. In case of the local bus error, the read command generates the exception with information about the address that couldn't be read.

4.3 Read-modify-write command (RMW)

This command accepts the following arguments:

Address - Address the of word which should be modified

Operation - Code of the operation - Increment, Decrement, Add, Subtract, And, Or, Xor

Operation dependent argument (not used in case of Increment and Decrement operations)

This command writes to the output buffer the original value of the modified word. In case of the local bus error, the RMW command generates the exception with information about the address that couldn't be modified, information whether the read or write access failed, and the original and final values of the modified word.

4.4 Read and test command

This command reads the word from the given address, and checks if it meets the defined condition. This command accepts the following arguments:

Address - Address of the word which should be tested

Operation - Code of the operation: Signed Less Than, Unsigned Less Than, Signed Greater Than, Unsigned Greater Than, And with mask and compare, Or with mask and compare

Operation dependent arguments - In case of the "Less Than" and " Greater Than" operations, this is just a value with which the word should be compared. In case of the "And with mask and compare" and "Or with mask and compare" operations, there are two arguments: the mask and the required final value.

In the case of the local bus error, the command generates the exception with information about the address that couldn't be read. This operation does not modify the register read. If the final calculated value does not pass the test, the command generates another exception with the information about the failed test operation and its arguments, and the original value that was read from the register.

4.5 Multiple read and test command

This command allows to perform automatically the test multiple times (e.g. waiting, until the condition is matched). The user can define how many times the test should be repeated, and what should be the delay between consecutive trials. The command accepts the same arguments as the previous one, preceded by two additional arguments:

Number of retries - Defines how many times the test should be retried before it is considered as failed.

Interval between retries - Defines the delay between repetitions of the test.

If the test succeeds, the status information is written to the output buffer, including: the address of the tested word, the number of retries performed before the test conditions were met, and the value of the word read in the final successful retry. If the test fails, the command generates the similar exception as the "Read and test command".

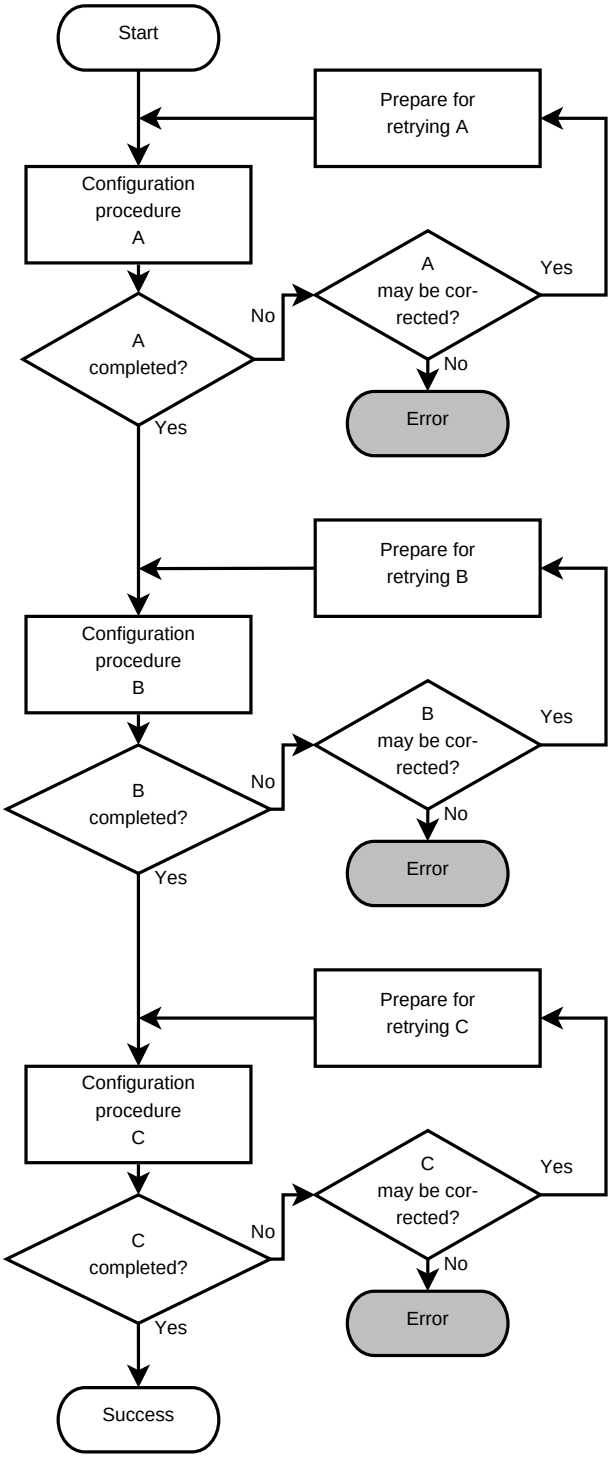
5. SOFTWARE SUPPORT

Even very efficient controller will be not accepted if it requires programming of control algorithms in a complex and unnatural way. Therefore, an important part of the concept is that the software routines may be written in a manner resembling controlling of the hardware via standard "old type" interface. Probably the approach presented in this section is only a one of multiple possible solutions. Maybe it is even not the most efficient solution. However, the purpose of this section is to show, that the controller described in Section 2 may be used to implement the control algorithm in a convenient manner.

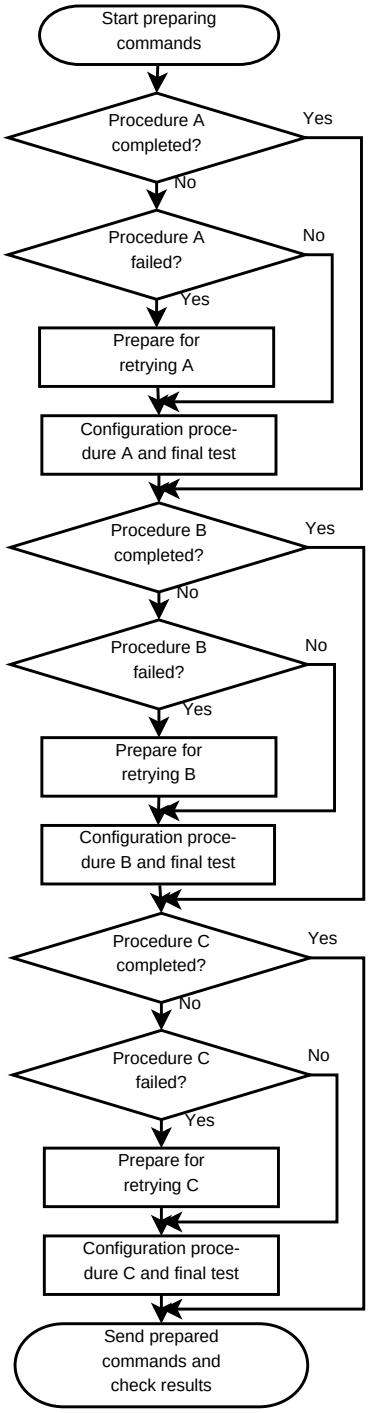
When working with a "standard" interface, the user simply sends commands and immediately verifies their results, retrying if necessary, or returning an error status (exception). With the proposed controller the problem is, that the information about the failure is delivered after a certain delay when commands related to the next part of the algorithm are already transferred for execution.

To compare coding of the control algorithm using the standard method and the new one, let us assume, that the control procedure is performed in three steps: A, B and C. After each step the handshake operation is performed to check if this step is successfully completed. In the standard implementation, where we assume that results can be read immediately after the operation is performed, the software will be usually implemented as shown in Figure 2a.

With the controller described in Section 4, the software should operate as shown in Figure 2b. The main difference is, that the software now does not execute commands directly but prepares a list of commands to be executed. After the complete list is sent for execution, the status is received and checked. The code organization shown in Figure 2b is just a proposal. Its main advantage is the similarity to the standard coding, shown in Figure 2a.



(a) Flow diagram of an algorithm implemented with standard method.



(b) Flow diagram of an algorithm implemented with the new, proposed method.

Figure 2: Comparison of control flow using the standard method, where results of the operation are available before the next operation is ordered and using the proposed method, where status is available after certain delay.

For each stage of the procedure, there are two flags provided. The first one informs if the stage is successfully completed. The second one informs if that stage has failed during the last execution (so the recovery procedure is needed). Initially, we set all flags to "false" (not completed and not failed). The software then generates commands, which perform each stage and then test if the particular stage was completed. The generated commands are then sent for execution, and afterwards the status is received. The software then analyzes the received status. If the status is "success", it means that the procedure was fully completed. If not, the software checks the reported exception (see Section 4). If it suggests, that the procedure may be retried, the software sets above flags appropriately and repeats the described procedure. During the repeated execution generation, the updated flags ensure skipping generation of commands, which implement completed stages. Similarly the updated flags ensure generation of commands, which implement necessary recovery procedures for failed stages. The whole cycle will be repeated until the entire procedure is successfully completed, or until the received status informs, that it cannot be retried due to a fatal error. Please note, that the round-trip latency of the interface delays only the waiting for the status of execution of the whole list, one time in each cycle.

6. RESULTS AND DISCUSSION

The concept presented in the paper is is not fully tested in the hardware yet. However, some of the components have been implemented and successfully used. For example the minimal controller with support for "read and test" and "multiple read and test" operations is successfully used in the "RPC Link Box Control System" developed for the CMS experiment at CERN.[11,12] In that application, it reads commands and data from a FLASH memory, not from a communication interface. However, this design has proven that such controller may be implemented with the minimal usage of logic resources in the FPGA. Other parts of the system have been tested in simulations. The preliminary results suggest that the described concept may significantly improve the speed of execution of control procedures in FPGA-based systems connected via modern communication interfaces.

7. CONCLUSIONS

The concept presented in this paper allows to better utilize high throughput of modern communication interfaces used to control FPGA based systems. The negative impact of high round-trip latency is mitigated thanks to the usage of a dedicated controller implemented in the FPGA. The controller autonomously performs the control algorithms together with handshake tests at the full speed of the local bus, and sends the final status of the entire set of commands. The minimalistic functionality of the controller keeps the resource usage at the reasonable level. The method to develop software routines cooperating with the proposed controller has been presented. The proposed method allows to code control algorithms in a manner very similar to the traditional approach, which should facilitate adoption of the new controller. However, presented results are still preliminary and not fully tested in hardware. Therefore, the goal of this article is to present the idea of the new controller, and submit it to the discussion. Further work is planned to prepare and test hardware implementations of the proposed controller together with supporting software.

ACKNOWLEDGMENTS

The author would like to thank Dr W.F.J. Müller from GSI for discussions and suggestions related to the idea presented in this paper.

REFERENCES

[1] "Understanding performance of PCI Express systems." `http://www.xilinx.com/support/documentation/white_papers/wp350.pdf`.

[2] Miller, D. J., Watts, P. M., and Moore, A. W., "Motivating future interconnects: A differential measurement analysis of PCI latency," in [*Proceedings of the 5th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*], *ANCS '09*, 94–103, ACM, New York, NY, USA (2009).

[3] Larrea, C. G., Harder, K., Newbold, D., Sankey, D., Rose, A., Thea, A., and Williams, T., "IPbus: a flexible ethernet-based control system for xTCA hardware," *Journal of Instrumentation* **10**(02), C02019 (2015).

[4] "GNU Radio, timing latency questions." `https://gnuradio.org/redmine/projects/gnuradio/wiki/UsrpFAQLatency` (12 2012).

[5] "What is the minimum latency of USB 3.0." http://stackoverflow.com/questions/13831008/what-is-the-minimum-latency-of-usb-3-0 (12 2012).
[6] Drozdov, I. and Zabołotny, W. M., "Versatile method to increase speed of external control with scatter-gather method in peripheral device." This Volume.
[7] PCI-SIG, "PCI Express base specification revision 3.1." https://members.pcisig.com/wg/PCI-SIG/document/download/8257 (2014).
[8] "RapidIO specification revision 3.1." http://www.rapidio.org/wp-content/uploads/2014/10/RapidIO-3.1-Specification.pdf (10 2014).
[9] Zabołotny, W. M., "Jam STAPL Player extensions for SCANSTA111 and fast transfer via VME/USB." post to alt.sources Usenet group, archived at https://groups.google.com/forum/#!topic/alt.sources/FHSbDDCcRos (2007).
[10] Mueller, W. F. J., "PDP-11/70 CPU core and SoC." http://opencores.org/project,w11.
[11] Zabolotny, W. M., Kudla, I. M., Pozniak, K. T., Bunkowski, K., Kierzkowski, K., Wrochna, G., and Krolikowski, J., "Radiation tolerant design of RLBCS system for RPC detector in LHC experiment," *Proc. SPIE* **5948**, 59481E–59481E–8 (2005).
[12] Zabolotny, W. M., Kudla, I. M., Pozniak, K. T., Kierzkowski, K., Pietrusinski, M., Wrochna, G., and Krolikowski, J., "RPC link box control system for RPC detector in LHC experiment," *Proc. SPIE* **5775**, 131–138 (2005).