

Interconnection System for the NetCOPE Platform

Tomáš Martínek and Jiří Tobola

12.12.2006

1 Abstract

This technical report describes the interconnection system for the general platform for rapid development of network applications (NetCOPE). The main objective of this interconnection system is to realize effective packet data transfers between components placed in FPGA and the host RAM memory. The basic parts of the interconnection system are internal bus, local bus, control bus and programmable DMA controller. The report describes in detail the architecture of these buses and shows how the programmable DMA controller works during packet reception and transmission.

Keywords: Platform for Network Applications, Internal Bus Architecture, Programmable DMA Controller, FPGA, PCI-X, PCI-Express, PowerPC

2 Introduction

Within the Liberouter project, several network applications have been already built based on the family of COMBO cards, such as the network interface card (NIC), NIC with hardware filtering and forwarding, the FlowMon probe [Zl05], Scampi monitoring adaptor, Intrusion Detection System [Kkh06] etc. The development of these applications was usually carried out separately and the projects shared only few common components. With the rising number of projects under development we have realized that it will be useful to share not only the common components, but the entire development framework. For these reasons, we embarked on designing a high performance general platform for rapid development of network applications called NetCOPE (Network Combo Pipe) with the following features:

- Network interface blocks for packet reception and transmission compliant with the IEEE 802.3 standard

- High throughput internal bus dedicated to packet transfers between FPGA and host RAM
- Access to the PCI-X and PCI Express buses
- Programmable bus Master DMA controller implemented using the PowerPC processor
- Efficient generic inter-component protocol with variable data width
- Set of generic IP Cores for packet analysis, classification, modification, timestamp processing etc.

With the NetCOPE platform, a designer implements only the application logic such as IDS or FlowMon probe and doesn't care about network input and output blocks, data transfer to host PC etc. He can also utilize available IP cores (Header-Field-Extractor processor, Look-Up processor, Output Packet Editor, etc.) to further shorten the development cycle. The block diagram of NetCOPE platform is shown in Figure 1.

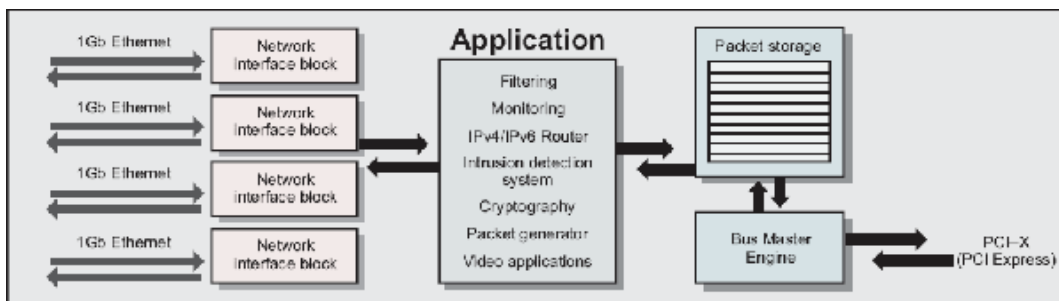


Figure 1: NetCOPE architecture

One of the most important parts of the NetCOPE platform is the *internal bus system*. The basic requirements for such a bus system include:

1. High throughput internal bus for transferring packet data between internal NetCOPE components and host RAM memory. This requirement is important especially for multi-gigabit network applications.
2. Easy connectivity of the system to the PCI host interface (PCI, PCI-X and PCI-Express)
3. Programmable DMA controller for effective control of DMA operations between the FPGA adaptor and host RAM. Programmability is very important as various network application have different demands for DMA transfers.

This technical report describes the architecture of the interconnection infrastructure in some detail. The system consists of three types of buses – (1) high throughput *internal bus*, (2) low speed *local bus* and (3) *control bus* dedicated for control messages between NetCOPE components and the *programmable DMA controller* that is used for efficient controlling of DMA operations.

This report is organized as follows: Section 3 shows the basic overview of NetCOPE interconnection system. The following sections then deal with the individual components – internal, local and control buses and the programmable DMA controller is shown. Conclusions are drawn in Section 8.

3 Overview of Internal Bus System

The NetCOPE platform covers three different types of buses (see Figure 2). The most important one is the *internal bus*. It is a high throughput bus designed for transferring huge amounts of data between the FPGA and the low level software driver. In other words, the internal bus connects components that require high throughput (e.g., Software Receive/Transmit Buffers, DRAM controller, potentially PowerPC processors etc.) with the the host PCI interface (PCI, PCI-X or PCI-Express).

The components, that don't require high bandwidth, can be connected using the *local bus*. Typically, the local bus is used for transferring configuration data, programs for microcontrollers, debug/status information, etc. The local bus is connected to the internal bus using the *LB bridge* component.

The last one is the *control bus*, that is reserved for of control data transfers between the *programmable DMA controller (PDMA)* and the components that require sending or receiving data to or from the host RAM memory. The PDMA controller is a key component that controls all DMA operations. It typically downloads *scatter-gather (SG)* lists from host RAM, performs necessary DMA transfers and uploads modified SG lists back to the software driver. As the programmability of DMA operation is an important feature of NetCOPE platform, we utilize the embedded PowerPC [Xrg03] processor as the main component of PDMA controller.

The following sections contain detailed information about each bus type.

4 Internal Bus

The *internal bus* is a high throughput bus dedicated for transferring of data internally between FPGA components as well as between FPGA components and the PCI interface. As shown in Figure 3, the internal bus utilises a tree

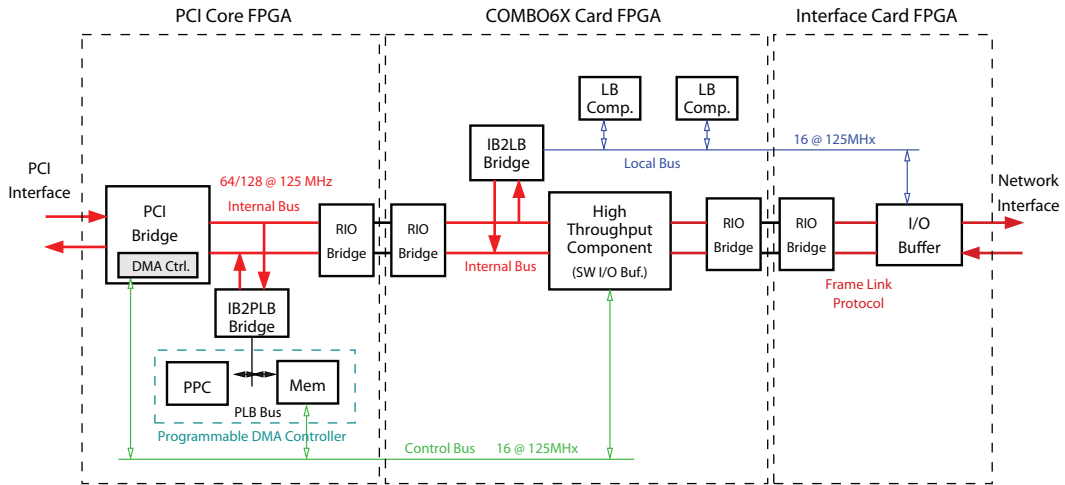


Figure 2: NetCOPE internal bus system architecture

structure composed of *root*, *switches* and *endpoints*. The root component is a part of the *PCI bridge* that provides communication with host PCI system. The switches control communication inside the internal bus and endpoints usually connect components that require huge data transfers to or from the software driver. Typical examples of components connected to endpoints are: software receive/transmit buffers, DRAM controller, internal to local bus bridge, potentially the PowerPC processor farm [Xpbg05] etc.

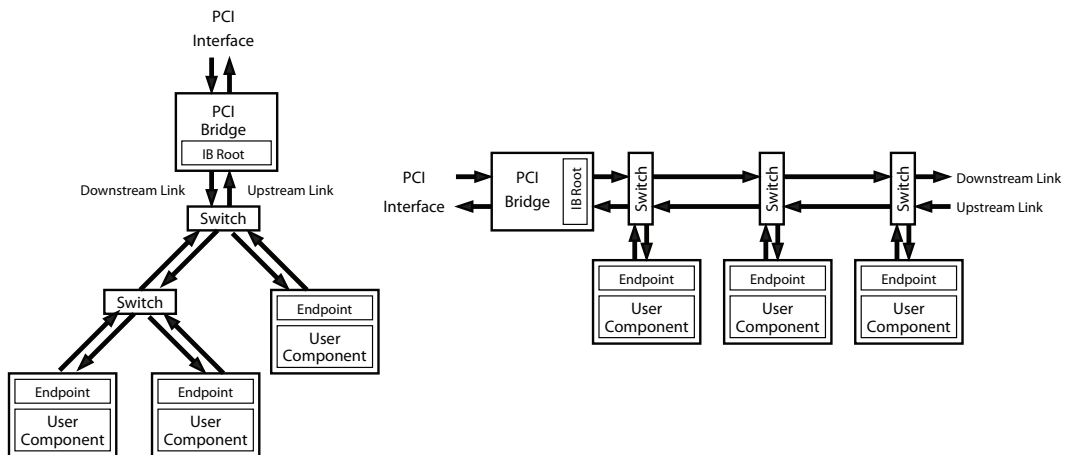


Figure 3: Internal Bus architecture

Depending on the configuration, the width of an internal bus link is either 64 or 128 bits and its clock rate is 125 MHz. Each link is full duplex so that data can be transferred simultaneously in both directions. This feature is very useful for point-to-point network application cores and allows for an easy connection to the PCI-Express host interface [Abs03].

The tree topology enables the endpoints to communicate with each other in separate branches without wasting the bandwidth of the global internal bus. As shown in Figure 3, the tree architecture can also be redrawn as a bus architecture with pipeline stages in the form of switch components. This kind of pipelining is very important in the FPGA context, because such wide buses are usually sensitive to distance due to limited FPGA wire routing resources.

The internal bus uses a packet-based communication protocol. Each packet consists of a header with necessarily control information and packet data. The timing diagram of the communication protocol is depicted in Figure 4. The packets are marked by the Start-Of-Packet (SOP) and End-Of-Packet (EOP) signals. Packet data transfer is controlled by means of the Source Ready (SRC_RDY) and Destination Ready (DST_RDY) signals. Using these two signals, the communication can be easily stopped by either the receiver or the transmitter. One of the main advantages of the internal bus communication protocol is that in the basic mode this protocol is compatible with FrameLink and Xilinx Local Link protocols. Due to this compatibility, it is possible to connect some of Xilinx IP Cores directly to internal bus infrastructure. The Aurora IP Core that provides communication via multigigabit transceivers is an example of such a component.

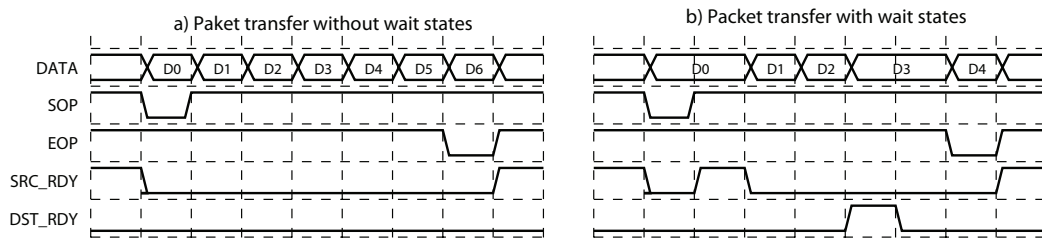


Figure 4: Internal bus communication protocol

5 Local Bus

The components that send or receive moderate amounts of data can be connected using the *local bus*. Typically, the local bus is used for transferring configuration data, programs for microcontrollers, debug/status information etc. As shown in Figure 5 the local bus has again a tree structure with main components being *root*, *switch* and *endpoint*. In comparison to the internal bus the local bus is much simpler. The communication can be initiated only by root, all endpoints work as slave nodes and the switch component simply forwards transactions from upstream ports to all downstream ports without any routing mechanism. Like in the case of the internal bus switch component, the local bus switch components represent pipeline stages to reduce the sensitivity to distance.

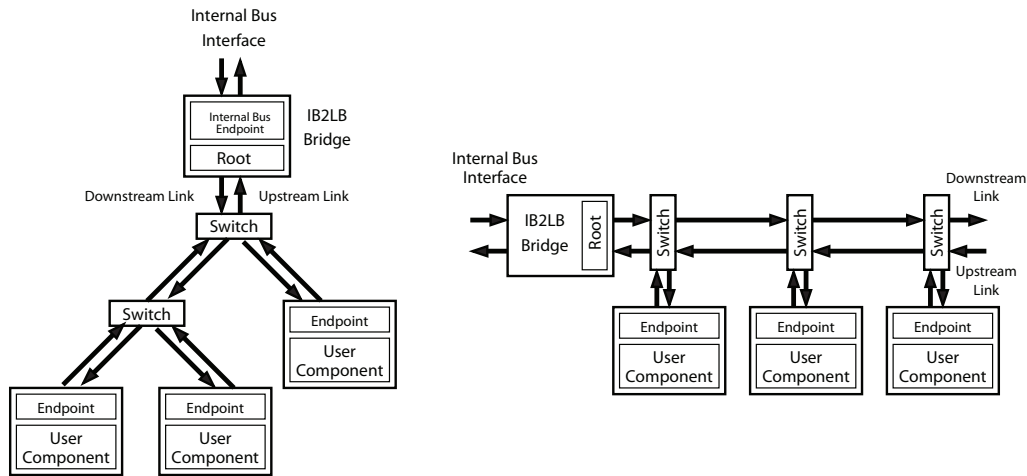


Figure 5: Local bus architecture

A local bus link is 16 bits wide and operates at 125 MHz. Each link is bidirectional, but the communication is usually performed only in one direction. (Bidirectional links are needed to avoid tri-state buses.) The communication protocol is shown in Figure 6. It is a simple protocol that distinguishes read and write transactions. Each transaction starts with an address followed by the data to be read or written with appropriate control signals. One of the main advantages of this communication protocol is that it can be easily transferred between multiple FPGA chips using just auxiliary registers – no dedicated bridge component is needed. The inter-FPGA communication only affects the latency of read transactions.

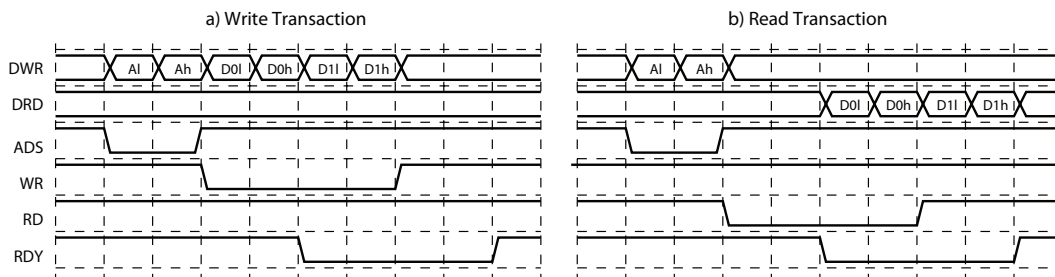


Figure 6: Local bus communication protocol

6 Control Bus

The *control bus* is reserved for control data transfers between the *programmable DMA controller (PDMA)* and components that need to send or receive data to/from the host RAM memory. Typical examples of such components are

software receive/transmit buffers, DRAM controller, etc. Like the internal and local buses, the control bus is based on a tree structure (see Figure 7). The root component is a part of the PDMA controller, the switches simply forward transactions from upstream port to all downstream ports and the endpoints connect the components that require DMA operations.

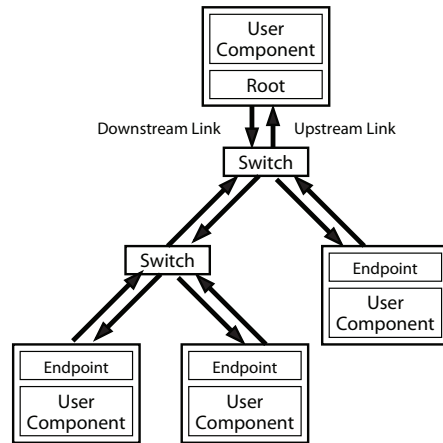


Figure 7: Control bus architecture

A control bus link is full duplex, 16 bits wide and operates at 125 MHz synchronously with other parts of the NetCOPE bus system. Transactions are transferred as packets containing headers and data. The communication protocol is compatible with FrameLink. As shown in Figure 8 a packet is marked by the Start-Of-Packet (SOP) and End-Of-Packet (EOP) signals. Data are transferred using the Source (SRC_RDY) and Destination Ready (DST_RDY) signals. The root component can send a packet to any of the endpoints and any endpoint can send a packet to the root. However, two endpoints are not allowed to exchange packets.

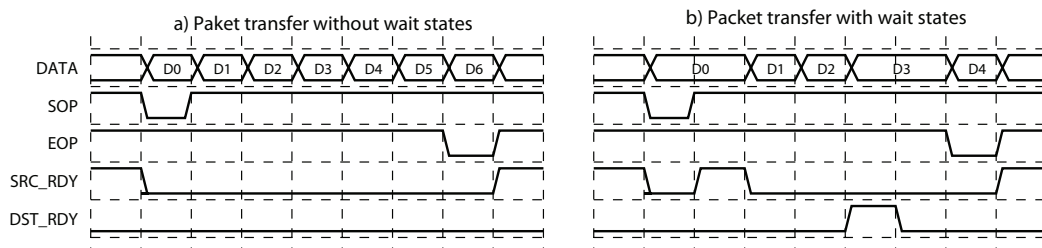


Figure 8: Control bus communication protocol

The root is a key component of the control bus system. As shown in Figure 9, the root architecture consists of sixteen receive and transmit queues, control registers and a small controller. All incoming messages from endpoints are split

into sixteen receive (RX) queues based on the endpoint identification. (Note: control bus supports only sixteen endpoints.)

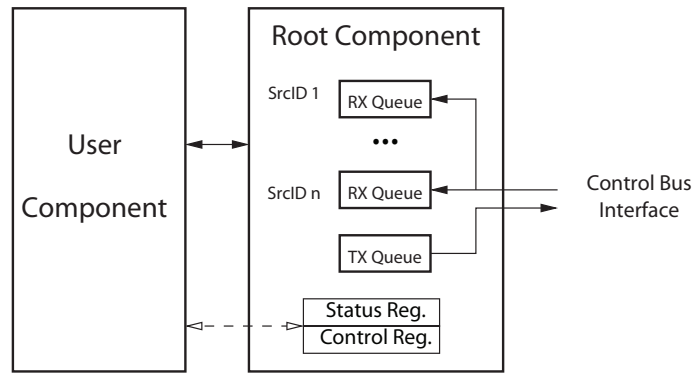


Figure 9: Root component architecture

All RX queues are implemented inside two-port embedded BlockRAM memory blocks. The first port is reserved for packets coming from the control bus. The second port is accessible from the user component that utilizes the root of the control bus. (Note: In our case the root component is utilized by the PDMA controller or the PowerPC processor). Start and end pointers of individual RX queues are stored in the root's status registers. The contents of the appropriate RX queue are available via a common memory interface. Finally, the end pointer can be moved by changing root's control registers.

Packet reception process operates in the following steps:

1. The packet coming from the control bus interface is stored into the appropriate RX queue based at the source address (endpoint identification).
2. The number of items in the RX queue as well as the RX queue start pointer are updated in the array of root's status registers.
3. The user component reads the content of the packet via direct RX queue memory access.
4. As the data are read, the user component writes the number of read items into the dedicated root's control register and the RX queue end pointer is moved to the appropriate position.

TX queues are realized in the same manner as RX queues. The root component contains sixteen TX queues; each of them is reserved for the appropriate endpoint. If the user component needs to send a packet from root to an endpoint, it simply writes the content of the packet into the TX queue memory and uses the root's status/control registers for packet transmission.

Packet transmission process operates in the following steps:

1. The user component reads root's status registers to obtain the TX queue start pointer
2. The user component writes the packet content into the appropriate memory position.
3. The user component writes the number of written items into the dedicated root's control register.
4. The root controller reads the required number of items from TX queues and sends them as a packet to the specified endpoint.
5. After the packet transmission, TX queue end pointer is updated.

7 Programmable DMA Controller

The main goal of the programmable DMA controller (PDMA) is to manage all DMA operations between the FPGA adaptor and host RAM memory. Typically, PDMA operates in the following steps:

1. Download scatter-gather lists (SG) into its local memory
2. Process all items in the lists, where each item represents a DMA operation
3. Modify appropriate parameters of SG lists
4. Finally, upload modified SG lists back to host RAM and possibly generate an interrupt.

The programmability of the DMA controller is a very important feature, because each application has different requirements and parameters relevant to its DMA operations. As Figure 10 shows, the architecture of PDMA is composed of a PowerPC processor (PPC), control bus root component and local memories for a PPC program and data. CB root's TX and RX queues are mapped to the PPC prefetchable PLB address space. Using this connection, all TX and RX messages can be processed very quickly inside internal PPC cache. Then, CB root's status and control registers are mapped into OCM interface.

As was mentioned in Section 6, all communication between PDMA and control bus endpoint components is based on sending messages. For example, if the new packet arrives into the system, the message is send to PDMA. Similarly, if a packet is processed the acknowledge message is sent from PDMA to endpoint.

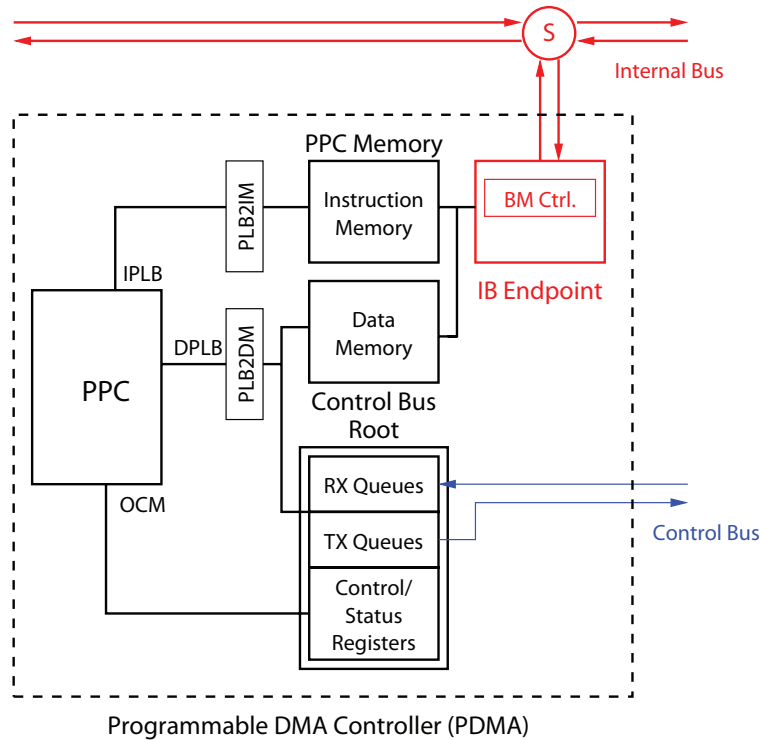


Figure 10: PDMA architecture

The previous implementation of PDMA was based on direct reading instead of sending messages. However, direct reading suffers very high latencies for read operations, which is not acceptable for high speed network application based on 10Gigabit Ethernet. In the next part, we will show examples how the PDMA works during the packet reception and transmission process.

The packet transmission process operates in the following steps:

1. PPC sends a request to a DMA controller, such as the controller placed inside the PCI bridge, to transfer the SG list from host RAM into the local PPC data memory.
2. As soon as the transfer is finished, the DMA controller sends an acknowledgement to PPC. (Note: PPC polls the RX queue status register for information about new incoming messages.)
3. PPC processes each SG list item in the following way:
 - (a) PPC sends a request to DMA controller to transfer the packet from host RAM into the internal buffer of the software transmit buffer (SW_TXBUF) component.

- (b) As soon as the transfer is finished, PPC sends a message to SW_TXBUF containing information about packet offset (inside internal SW_TXBUF buffer), packet length and potential flags.
 - (c) SW_TXBUF forwards the required packet to network interface and sends an acknowledgement message back to PPC.
4. As soon as all items of the SG list are processed, PPC sends a request for downloading a new SG list.

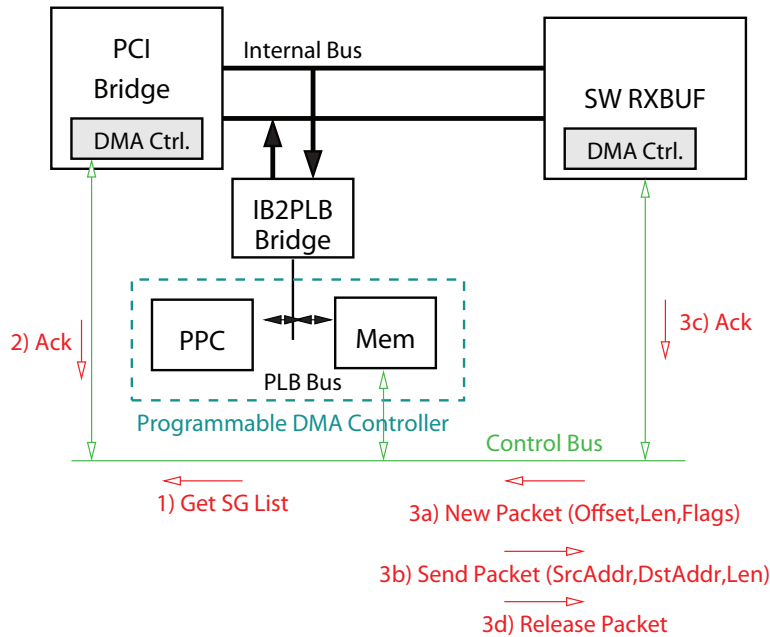


Figure 11: Example packet transmission process

Packet reception process operates in the following steps:

1. PPC sends a request to a DMA controller, such as the controller placed inside the PCI Bridge) to transfer the SG list from host RAM into the local PPC data memory.
2. As soon as the transfer is finished, the DMA controller sends an acknowledgement message to PPC.
3. PPC processes each SG list item in the following way:
 - (a) If a new packet arrives into the system, the software receive buffer (SW_RXBUF) generates a message to PPC containing information about the packet offset, packet length and potential flags.

- (b) PPC sends request to the DMA controller to transfer the packet from SW_RXBUF component to host RAM.
 - (c) As soon as the transfer is finished, the DMA controller sends an acknowledgement message back to the PPC processor.
 - (d) PPC sends an acknowledgement message to the SW_RXBUF component and the incoming packet can be released from the SW_RXBUF internal buffer. Concurrently, PPC modifies the appropriate parameters or flags (e.g., interface number, packet error status etc.) in the SG list item.
4. As soon as all items of SG list have been processed, PPC sends a request to the DMA controller for uploading the modified SG list back to host RAM and downloading a new SG list to the PPC local data memory.

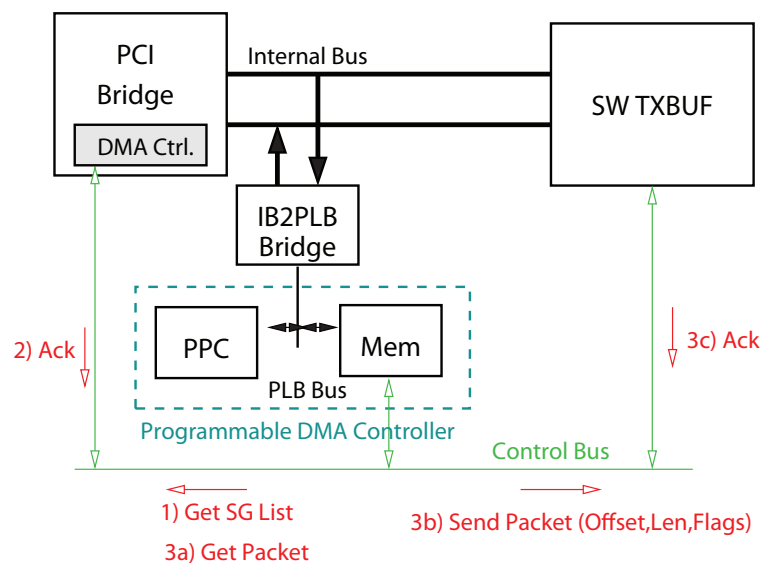


Figure 12: Example packet receiving process

Finally, an example of the physical connection between the SW_RXBUF and SW_TXBUF components is shown in Figure 13. PPC uses two DMA controllers. The first one is placed inside an PDMA internal bus endpoint and controls all DMA operations between the PDMA local memory and host RAM. The second one is placed inside the SW_RXBUF and SW_TXBUF components and controls all DMA operation between SW_TXBUF, SW_RXBUF components and host RAM. Both DMA controllers are connected using control bus endpoints and can be controlled via two TX and RX queues of PDMA. Two other control bus endpoints are used for communication with SW_RXBUF and SW_TXBUF components, so the next to RX and TX queues are reserved for them.

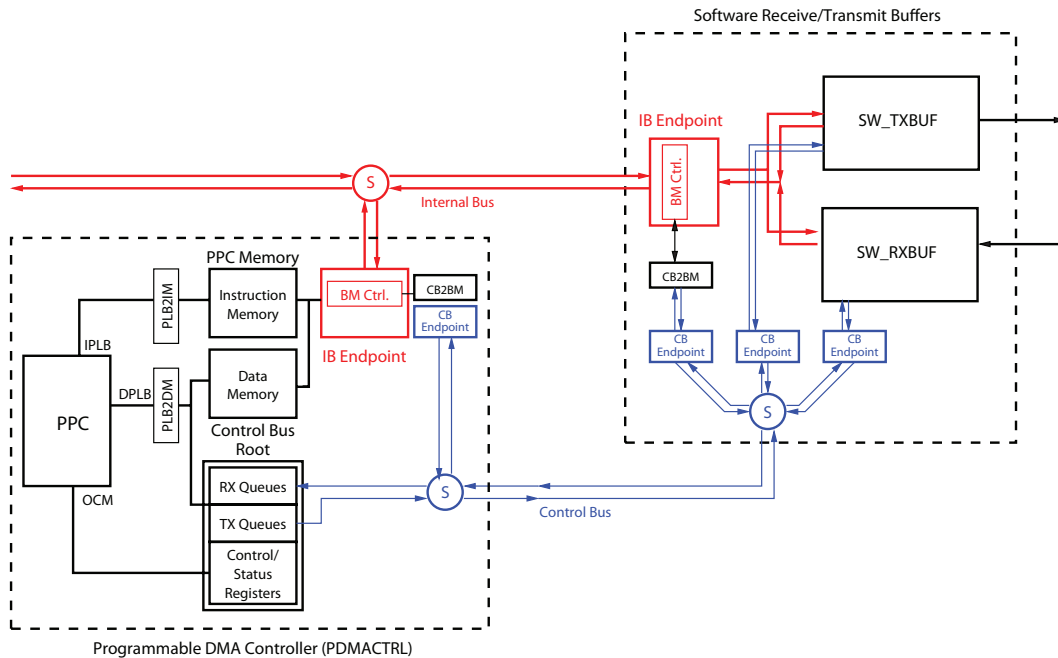


Figure 13: Example of PDMA connection

8 Conclusions

This report proposes the architecture of the NetCOPE interconnection system. We described the internal, local and control buses together with their communication protocols. Moreover, the programmable DMA controller (PDMA) was presented and its functionality illustrated by examples of packet reception and transmission.

References

- [ZI05] Žádník M., Lhotka, L.: *Hardware-Accelerated NetFlow Probe*, Technical Report 32/2005, CESNET, Praha, 2005.
- [Kkh06] Kobierský P., Kořenek J., Hank A.: *Traffic Scanner*, Technical Report 33/2006¹, CESNET, Praha, 2006.
- [Tk06] Tobola J., Košek M.: *Frame Link Tools*, Technical Report, in preparation, CESNET, Praha, 2006.
- [Abs03] Anderson D., Budruk R., Shanley, T.: *PCI Express System Architecture*, MindShare, Inc., September 4, 2003

¹<http://www.cesnet.cz/doc/techzpravy/2006/ids/>

- [Xrg03] Xilinx, Inc.: *PowerPC Processor Reference Guide*, September, 2003
- [Xpbg05] Xilinx, Inc.: *PowerPC 405 Processor Block Reference Guide*, July, 2005