Perfection in Automation
www.br-automation.com

POWERLINK IP-Core
Generic Documentation

Date: July 30, 2012

Project Number: AT-B0-000002

I Versions

Version	Date	Comment	Edited by
0.1	Jul 30, 2012	First Edition	Joerg Zelenka
0.2	Dec 7, 2010	Added Asynchronous 8/16bit Parallel Interface Added openMAC IP-core Added feature to SPI Added system description information Omit PDO Descriptors	Zelenka Joerg
0.3	Jan 10, 2011	Extend documentation of openMAC Added wake up functionality	Zelenka Joerg
0.4	Mar 21, 2011	Extend documentation of openMAC Internal Bus Memory Mapped Master	Zelenka Joerg
0.5	May 6, 2011	Changes in 8/16bit Parallel interface	Zelenka Joerg
0.6	Aug 1, 2011	Added setup/hold time for 8/16bit Parallel interface	Zelenka Joerg
0.7	Sep 7, 2011	Revised openMAC section (renamed to openMAC Ethernet) Added description of new Internal Bus Memory Mapped Master	Zelenka Joerg
0.8	Nov 21, 2011	Added Time Synchronization feature Added PDI Interface definition	Zelenka Joerg
0.9	Nov 29, 2011	Added openMAC DMA observer feature	Zelenka Joerg
1.0	Dec 14, 2011	Revised documentation Converted to official POWERLINK IP-Core Generic Documentation	Zelenka Joerg
1.1	Dec 20, 2011	Changed double buffer switch	Zelenka Joerg
1.2	Jan 11, 2012	Changed MAC Cmp register layout	Mair Thomas
1.3	Jul 30, 2012	Revised 8/16bit Parallel interface timing	Zelenka Joerg

Table 1: Versions

II Safety Notices

Safety notices in this document are organized as follows:

Safety notice	Description
Danger!	Disregarding the safety regulations and guidelines can be life-threatening.
Warning!	Disregarding the safety regulations and guidelines can result in severe injury or heavy damage to material.
Caution!	Disregarding the safety regulations and guidelines can result in injury or damage to material.
Information:	Important information used to prevent errors.
Example:	Functionality is described with an example to prevent errors.

Table 2: Safety notices

III Table of Contents

1 Introduction

This documentation introduces the functionality of the POWERLINK IP-Core without any FPGA-vendor specific definitions or considerations.

Please note that the generic documentation nature is indicated in the file name by the suffix "_Generic". If you require platform-dependent information, refer to the POWERLINK IP-Core documentation specific to the FPGA-vendors –refer to the suffix in the filename as well!

In Fig. 1 the POWERLINK IP-Core is visualized with its various configuration possibilities. The POWERLINK IP-Core includes the POWERLINK-specific MAC-layer components (openMAC and openHUB) and application interface IP-cores (Direct I/O and Process Data Interface).
Note that the POWERLINK IP-Core does not include the POWERLINK Communication Processor (PCP), as well as the Application Processor (AP), however, these can interface to the POWERLINK IP-Core via internal bus interfaces easily.

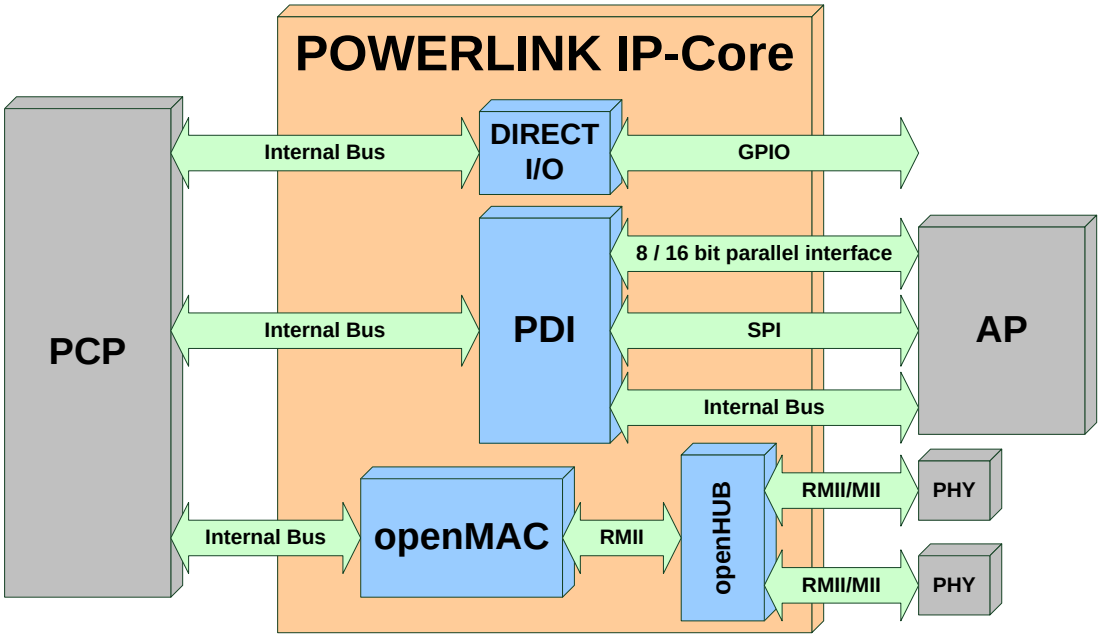

Fig. 1: POWERLINK IP-Core Block Diagram

2 Design Considerations

The POWERLINK IP-core has to encapsulate all necessary components to run a POWERLINK device on an FPGA. The POWERLINK IP-cores execute the following tasks:

- Communication over Ethernet (IEEE 802.3) with 100Mbps half-duplex
- POWERLINK specific hardware acceleration to serve highest performance (e.g. low response delay)
- Flexible network topology with two Ethernet ports
- Data interface for data exchange to application specific components
- Simple user configuration via GUI in FPGA development tools
- Sparse resources in FPGA

2.1 System Overview

The POWERLINK IP-core is built up depending on the configuration (generics). Independent of the configuration case the communication part of the POWERLINK IP is the same and consists of the following components:

- openMAC
- openHUB
- openFILTER
- DPRAM (packet buffer) or DMA (packet buffer not within the IP-core)
- Optional RMII to MII converter

Fig. 2 shows the Configuration Case using a Process Data Interface to an internal/external AP. The PDI includes a SYNC device that is responsible for the synchronization of the AP. It can generate IRQ by software or time-triggered by the openMAC. The last possibility is recommended for very low jitter synchronization tasks.
The data exchange between PCP and AP is done via the DPRAM. This memory type allows a simultaneous access to the content by the PCP respectively AP. The process data is exchanged via the TripleLogic additionally. This logic ensures that the consumer can access the most current and not locked data at the moment. “Locked data” means that the consumer is using this virtual buffer, thus the producer may not access.
The PCP respectively AP takes the role as consumer and producer depending on the type of process data – refer to Tab. 1.

Tab. 1: Producer/Consumer Definition

Process Data Objects	Producer	Consumer
RX	PCP	AP
TX	AP	PCP

Fig. 3 shows the second Configuration Case that instantiates an I/O Port device for simple I/O port applications. Beside the I/O registers – containing the input and output data – an I/O Latch is used. This allows inputting data by a strobe respectively validating output data by a valid flag.

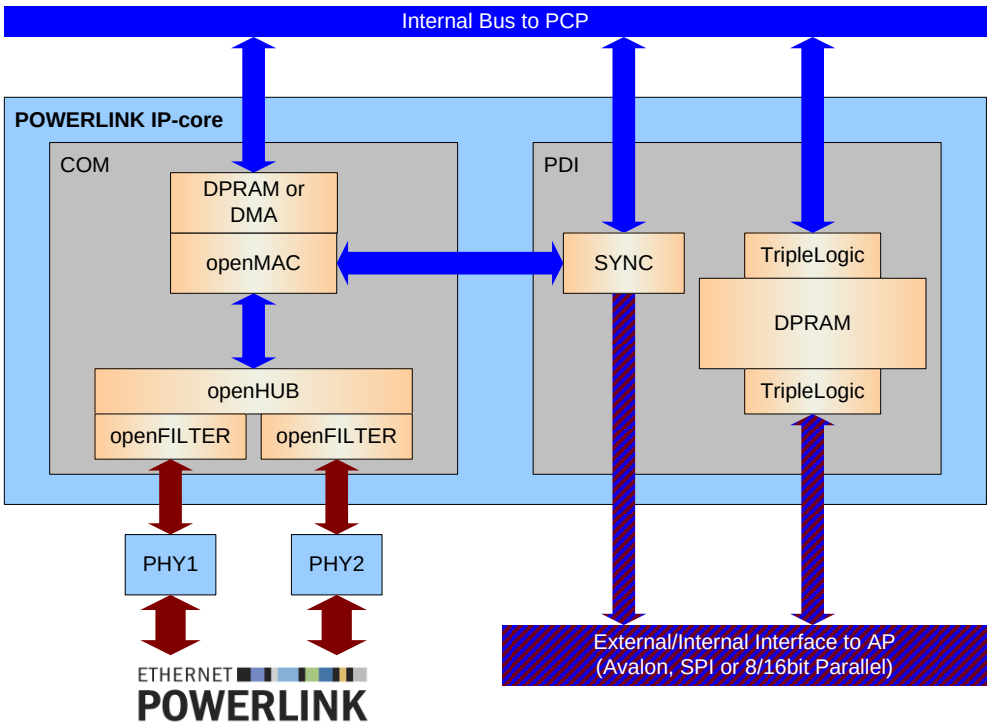

Fig. 2: POWERLINK IP-core System Overview Configuration Case 1

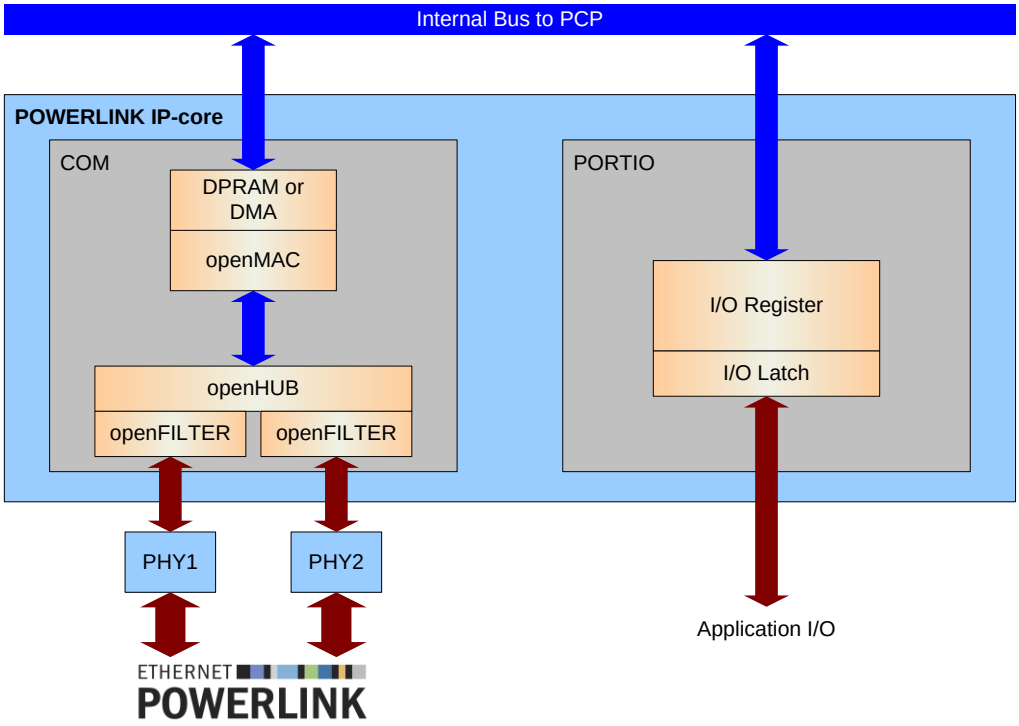

Fig. 3: POWERLINK IP-core Sysrem Overview Configuration Case 2

2.2 System Configuration

The POWERLINK IP-core can be configured to different architectures applicable for different applications and interface types:

Only PCP Configuration (Fig. 3)

- Port I/O
  - General Port I/O Interface
  - Port I/O Interface for ADC / DAC (using I/O Latch)

PCP + AP Configuration (Fig. 2)

- PDI with internal bus to AP
- PDI with external bus to AP
  - External MCU or DSP
  - 8/16bit parallel bus interface
  - SPI

Depending on the user's choice the GUI enables to insert different values to scale the POWERLINK IP:

- Number of RPDO (necessary for cross traffic ability)
- TPDO/RPDO size
- Asynchronous buffer size
- External parallel port data width
- SPI configuration (clock polarity/phase)

These configuration values are used to calculate all necessary generics for the POWERLINK IP-core and the instantiated components. For instance the number of RPDO is required to calculate the amount of receive buffers used by the openMAC.

Tab. 2: Available Configurations

Nr	Configuration	openMAC[1]	PDI	PORTIO
1	Simple I/O (no AP)	✓	×	✓
2	Internal AP	✓	✓[2]	×
3	External AP (e.g. MCU or DSP)	✓	✓[3]	×
4	openMAC only	✓	×	×

[1] OpenMAC is mandatory for every configuration.
[2] The Internal AP Interface uses an Internal Bus.
[3] The External AP Interface can be configured to use an 8/16bit parallel port or SPI.

3 IP-Core Architecture

This chapter describes the architecture and functionality of the IP-cores included in the POWERLINK IP-core package. The description should give an idea of the functionality and will not specify every single component or block in the HDL design. This approach enables hardware and software designers to develop further components/features, build test benches/cases and debug malfunctions.

3.1 POWERLINK

The POWERLINK IP core is simply the top level (powerlink.vhd) that interconnects all necessary components for a given application. For instance a simple I/O POWERLINK slave does not need a DPR for PDO exchange, since there is no AP present. In contrast a POWERLINK slave with an external AP connected via SPI will require other IPs than an internal AP.

The generation of the POWERLINK IP-core is controlled by generics, which enable or disable associated components for generation. Furthermore some minor signal assignments (e.g. high-active to low-active conversation) and clock synchronizations are done. Thus every single component (e.g. openMAC or PDI) can be used without the POWERLINK IP-core, what requires further considerations for configuration by the designer.

Tab. 2 gives an overview of the used components depending on the application. Fig. 2 and Fig. 3 show the architecture of the POWERLINK IP depending on the configuration.

3.1.1 Clock Sinks

The POWERLINK IP-core requires several specific clock signals described in Tab. 3. It is mandatory to connect clk50 with a 50MHz clock signal and clkEth with a 100MHz clock signal. Clk50 and clkEth must by synchronous.

Tab. 3: Clock signals

Clock name	Frequency [MHz]	Description
clk50	50	This clock must be driven by a 50MHz source. It is used for openMAC, openHUB, openFILTER and all components synchronous to the Ethernet interface.
clkEth	100	This clock is necessary if the IP-core is used in combination with RMII. The openMAC's TX-signals are latched with the falling edge. If MII is used, this clock signal is not connected. Note: It is recommended to use RMII!
m_clk	External memory controller	This clock drives the master logic for the packet transfer handling. It is highly recommended to connect this clock signal with the same that is used by the connected memory controller![4]
pkt_clk	Any	The packet clock signal is directly connected to the DPRAM of the packet buffer.
clkPcp	Any	The clkPcp signal can be driven by any frequency. clkPcp is connected to the PCP side of the PDI.
clkAp	Any	clkAp can be connected to a clock signal with any frequency.

3.2 OpenMAC Ethernet

The openMAC Ethernet IP-core (openMAC_Ethernet.vhd) is a top-level component that instantiates the necessary parts to create the MAC-layer. It includes the following components:

- openMAC (OpenMAC.vhd)
- Phy management (OpenMAC_PHYMI.vhd)

[4] This enables highest performance possible with the applied memory technology.

- Phy activity generator (OpenMAC_phyAct.vhd)
- openHUB (OpenHUB.vhd)
- openFILTER (OpenFILTER.vhd)
- RMII to MII converter (rmii2mii.vhd)
- openMAC timer compare unit (OpenMAC_cmp.vhd)
- Packet Buffer DPRAM (OpenMAC_DPR_*.vhd)
- openMAC DMA master for internal bus interface (openMAC_DMAmaster.vhd)
  - Memory Mapped Master handling logic for internal bus (master_handler.vhd)
  - openMAC DMA handling logic (dma_handler.vhd)
- minor library components (e.g. sync)

Overview

Fig. 4 shows the overview of the openMAC Ethernet IP-core configuration. The main part of this IP is the openMAC itself, a MAC with hardware accelerations designed for Industrial Ethernet's real-time requirements. Via an RMII the MAC is connected to a 3-port hub (openHUB) that allows two Ethernet phys for flexibility. In between an "anti-distortion-filter" (openFILTER) is used to lock out distortions on the network accessing the node. Since the filter is instantiated twice any disturbance is prevented from propagating around the network.
The openMAC itself provides a high-accurate timer, which is used for packet time stamps and forwarded to a timer-compare-unit (CMP). The received and to be transmitted packets are transferred in three different manners:

- RX and TX to/from Packet BUF
- TX from Packet BUF and RX via internal bus
- RX and TX via internal bus

In order to configure and monitor the external Ethernet phys the Serial Management Interface (SMI) is used. The "openMAC MII" component does the communication to every connected phy.

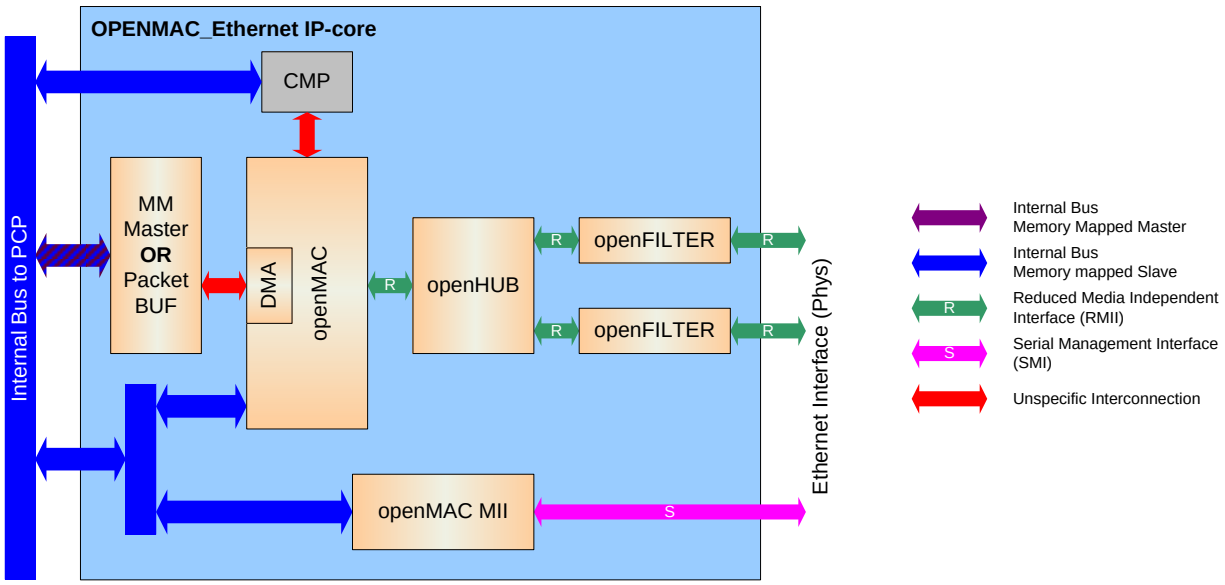

Fig. 4: OpenMAC IP-core – overview

3.2.1 MAC (openMAC)

The openMAC component provides a 100Mbps half-duplex Ethernet interface like a general MAC. Furthermore extra features are implemented to allow hard-real time possibility (Industrial Ethernet).

The following components are integrated to enable hard-real time applications:

- Ethernet RX Packet Filter (first 31 octets):
  OpenMAC provides 16 independent RX filters that observes the first 31 octets in any kind of Ethernet message. This allows a high flexibility of filter configuration.
- Auto-Response Ability starts TX after IPG[5] (adjustable):
  Every filter can start the transmission of a specific packet after the reception of a matched message.
- 32bit Timer for time-stamps (resolution 20ns):
  Every packet that is received and every packet that was transmitted gets a 32bit time-stamp.

Information:

In order to get a more detailed documentation about openMAC, openHUB, openMAC MII and openFILTER please refer to "openMAC & Components Documentation" (openMAC.pdf)!

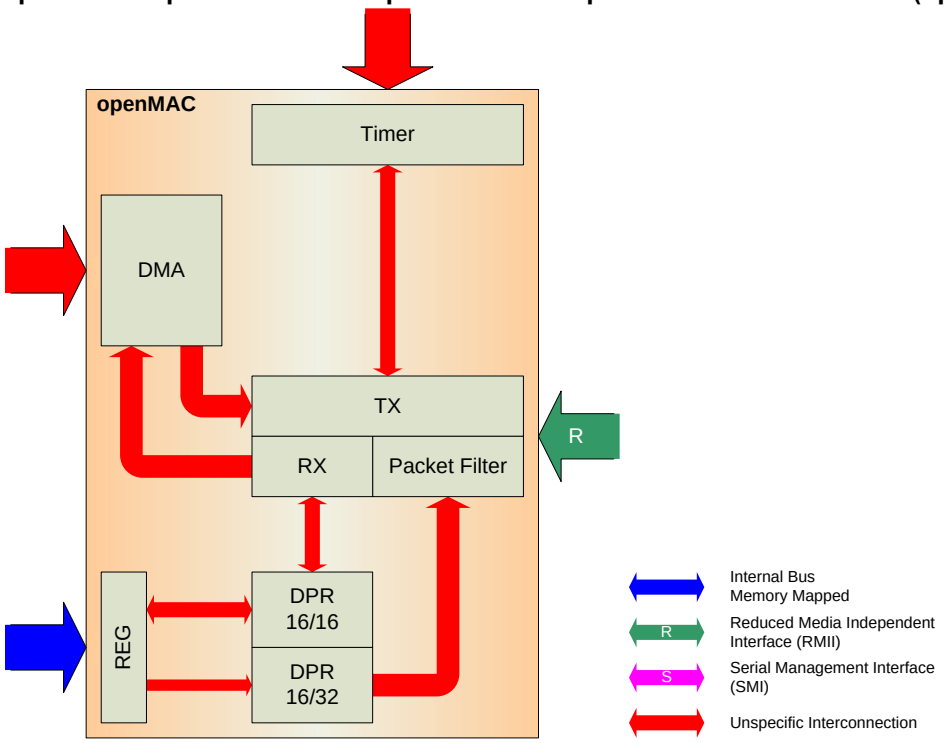

Fig. 5: OpenMAC Block diagram

3.2.1.1 DPR

The openMAC component includes two different DPR necessary for the RX Packet Filter and the descriptors (TX/RX). The filter patterns are stored in a DPR with different data width on the ports – the MAC-internal port is equipped with 32bit data width. The filter DPR's data flow is only unidirectional, thus the filter patterns set to the DPR are not readable.
The descriptor storage is done in a DPR with 16bit data width on both sides.
Both DPR are driven in only one clock domain of 50MHz (RMII Clock).

3.2.1.2 RX Packet Filter

OpenMAC provides 16 independent RX packet filters that observe the first 31 octets in an Ethernet packet. The observation is started after the SFD[6] beginning with the destination MAC address. Thus the filter can verify the MAC addresses (destination and source), the Ethernet Type/Length field and the first 17 bytes of the Ethernet frame payload.

[5] IPG: Inter Package Gap (960ns in case of 100Mbps)
[6] SFD: Start of Frame Delimiter

Filtering is done simultaneously by comparing the received data with the filters in the order starting at filter 0 to 15. When one enabled filter matches to the received data, the filter number is written to the used descriptor. If two or more enabled filters match to the RX data, the filter with the lowest number (zero to 15) will signalize the match event. If no filter matches to the received data, the MAC's DMA stops its transfer.

Information:

In order to clarify the RX flow the following list is introduced:

- **An Ethernet packet arrives with correct Preamble and SFD.**
- **The openMAC's DMA starts to transfer data to a free RX descriptor and filters simultaneously.**
- **After the reception of 31 octets the matching results are verified in the order of filter numbers.**
- **The matching filter with the lowest filter number is written to the used RX descriptor.**
- **The reception is cancelled if no filter matches.**

Since the matching information is known after receiving 31 octets, the MAC's DMA transfers at least 32 bytes (word transfers) of an out-filtered packet to the memory. This approach avoids the necessity of additional resources (packet buffering) and will not be recognizable by the PCP.

After a packet was completely received (enabled filter matches) and the CRC[7] is correct, an IRQ will be generated. When the filter is used for auto-response ability the IRQ will be generated as well.

3.2.1.3 Auto-Response Ability

In order to serve a fixed low-latency response every filter can start the transmission of a packet. The filter must be enabled and associated with a TX descriptor that point to a valid or addressable TX packet buffer. In addition the IPG can be increased for some ticks (32bit) the specific TX descriptor.

Example:

For filter number 4 the following assumption is taken:

- **Match with the POWERLINK frame PReq**
- **Enabled for auto-response**
- **Set to TX descriptor number 10**
- **Enabled**

For the associated descriptor number 10 the following assumption is taken:

- **Points to a TX buffer that stores the auto-response packet (PRes)**
- **The IPG is not increased**
- **Owner is the MAC (owner bit is set)**

When the MAC receives a PReq frame filter number 4 will signal a match. The PReq packet will be forwarded by the DMA. If the CRC is correct a free RX descriptor will point to the location of the stored PReq in the memory. After the IPG the MAC automatically starts the transmission of the associated packet (PRes, TX descriptor number 10).

3.2.1.4 Timer

The openMAC includes a free-running 32bit timer that is used for time stamp generation (for RX and TX frames). The RX time stamp is set by the MAC at the reception of the SFD. For TX packets the time stamp of the associated descriptor is set at the beginning of the preamble. If a collision occurs, the time stamp of the last try is stored in the descriptor.
The 32bit timer value is provided at the port map of the openMAC's entity (Mac_Zeit) as an output and can be used for time-triggered interrupts. The benefit of using the MAC's timer is to enable very low jitter

[7] CRC: Cyclic Redundancy Check (Ethernet uses 4 bytes, openMAC checks the CRC)

synchronization to the network. The time stamps of the received/transmitted packets can be used as a reference.

3.2.1.5 DMA

The openMAC's DMA is used to transfer RX/TX data. Since the implementation has no temporary buffer included, the data arrival latency must not exceed a certain limitation. Otherwise the current data is lost (RX) or the MAC sends old data (latched with DMA_Ack).
The MAC's DMA is connected to a simple master interface with the following signals:

- Dma_Req (Request)
- Dma_Rw (Write/Read Request)
- Dma_Ack (Acknowledge)
- Dma_Addr, Dma_Din and Dma_Dout
- Dma_Rd_Done and Dma_Wr_Done

The DMA starts the transfer by asserting Dma_Req and Dma_Rw ('1' = Read). For a write transfer the data is set synchronously to Dma_Dout. For a read transfer the slave must set the data to Dma_Din. When the slave asserts Dma_Ack the transfer is finished.
Fig. 6 shows the timing of a read transfer. It is important that the read data is set for at least one cycle after the assertion of Dma_Ack. The address is set synchronously with the read request.
Fig. 7 shows the write transfer timing. The DMA sets the valid data synchronously with the request and the associated address.
The signals Dma_Rd_Done and Dma_Wr_Done are asserted for one cycle to signalize that a write or read transfer is finished, which implicitly means that a RX or TX packet ends.

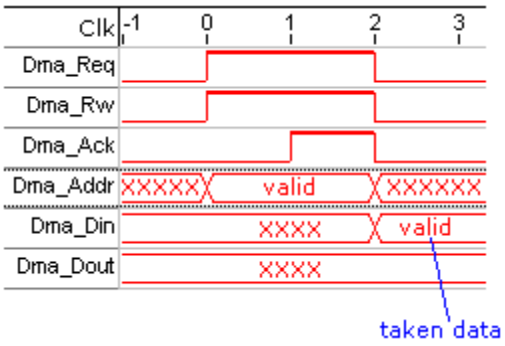

Fig. 6: DMA read transfer

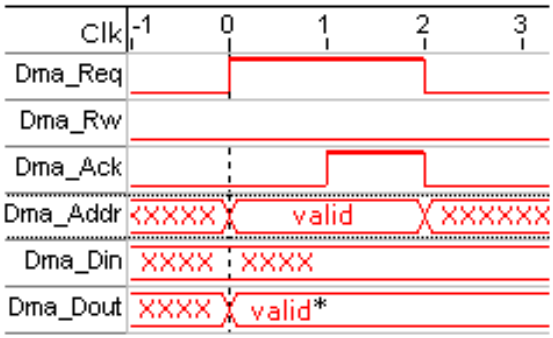

Fig. 7: DMA write transfer[8]

As already mentioned the DMA transfers must occur within a certain time limit.

Information:

When the MAC is configured to half-duplex, every 8th cycle a new DMA transfer is initiated.
In case of full-duplex mode every 4th cycle a new request will be asserted if TX and RX are active simultaneously.
If these limitations are not satisfied, corrupted data is transmitted to the network respectively the received data is incorrect!

3.2.1.5.1 Observer

In order to debug a failing DMA transfer the Dma_Req_Overflow output port is introduced, which is asserted by the DMA at the last possible acknowledge cycle. The DMA handler logic (dma_handler.vhd) combines the Dma_Req_Overflow with the Dma_Reg signal in order to detect a DMA transfer error. In case of an error a register is set, which can be accessed by the CPU for debugging purpose (refer to 4.2.2.2).

[8] DMA write transfer: Dma_Dout is valid until next Dma_Req assertion.

3.2.2 Packet Buffer (BUF)

In order to be independent of external memory types and the system's interconnection, an internal packet buffer is used. This allows low-latency access to the packet storage by the MAC, which is essential for TX packets.
The packet buffer is implemented as a dual-ported memory with independent clock sources at both ports. This allows simultaneous and low-latency access by the MAC and the PCP. Nevertheless the PCP must not access to packets that are locked by the associated descriptor (owner bit).
Due to the ability of independent clock sources at the two ports, the PCP can access with its defined clock rate without any synchronization latencies introduced.
The usage of the packet buffer is optional, depending on the generic set (useIntPacketBuf_g and userx-IntPacketBuf_g).

3.2.3 Internal Bus Memory Mapped Master (MAC_DMA)

The system designer has the possibility to store RX and/or TX packets in memory locations other than the provided dedicated packet buffer (BUF, 3.2.2). The reason for this is to reduce FPGA resource utilization (memory blocks), which enables smaller FPGA sizes. However, it has to be ensured that the memory's read latency is static, which is valid for FPGA-internal memory blocks and external SRAM devices. Furthermore, other master devices connected to the memory affect the access latency as well, which has to be considered in the design. The Internal Bus Memory Mapped Master logic is able to transfer data with single beat or burst transfers, which has to be defined by generics.

Important:

Only use the Internal Bus Memory Mapped Master interface if the connected memory is either FPGA-internal or external SRAM (with 10 ns speed grade). In addition the number of master devices connected to the shared memory and the arbitration algorithm has to be defined carefully. If the connected memory device is e.g. SDRAM, it is not recommended to store TX packets in that kind of memory.[9] RX packets may be stored in dynamic RAM types, however, the designer has to verify the system's stability.
Also note that the openMAC software driver implementation assumes that the packets are linked to the heap section of the system.

The Internal Bus Memory Mapped Master logic is implemented in the file openMAC_DMAmaster.vhd and is instantiated by the openMAC_Ethernet.vhd component depending on the generic set. The master and DMA handler logic is implemented in the master_handler.vhd and dma_handler.vhd file respectively.

3.2.3.1 Architecture Design

In order to achieve highest performance in the context of the FPGA design, the clock domain crossing is tightly coupled to the openMAC DMA component. Due to this approach the connected memory controller is allowed to be located in any different clock domain compared to the openMAC clock (RMII, 50 MHz), which eliminates the need of clock domain crossing logic or even clock crossing bridges.
The architectural design is visualized in Fig. 8, which assumes RX and TX packet transfers via the Internal Bus Memory Mapped Master logic (no packet buffer selected by the user, 3.2.2). The packet transfer handling is divided into two handlers, which are located in the respective clock domain. The DMA handler is coupled to the openMAC DMA interface, which performs with the RMII clock (50 MHz). The master handler is located in the other clock domain, which is coupled to the memory controller's clock signal. In order to transfer data to different clock domains two approaches are chosen:

- 2-stage synchronizer (SYNC)
- Dual clocked FIFO

[9] Dynamic RAM types have to refresh their data content, which introduces long access latency to the host (e.g. CPU or DMA).

The dual clocked FIFO is applied to the RX and TX packet data transfer, which enables highest throughput (without considering delays introduced by interconnect or memory) from/to the memory respectively. The 2-stage synchronizer are implemented with two chained flip-flops avoiding metastability. The SYNC instance is used to transfer the write/read base address from the openMAC DMA to the master handler. Furthermore, the DMA and master handler communication via the synchronizers.
Note that the necessary synchronization between the DMA and master handler introduces a certain delay (refer to 3.2.3.2), which only affects the beginning of the packet transfer. Afterwards the master handler tries to fill the TX FIFO to a certain limit, which ensures presence of TX data to openMAC (avoiding FIFO underflow). In the opposite direction (RX), the master handler tries to empty the RX FIFO continuously, which avoids FIFO overflow.

DMA handler

The DMA handler decodes the qualifiers asserted by the openMAC DMA (e.g. dma_req and dma_rw), and communicates them via the synchronizers to the master handler. In addition the handler controls the write port of the RX- and the read port of the TX FIFO. The handler is able to decode the very first DMA request (TX or RX respectively) and capture the openMAC's DMA address presented on its interface port. This value is forwarded via the synchronizers to the master handler.

Master handler

The master handler is a more complex logic compared to the DMA handler, however, the master handler complies to the Altera Avalon Interface Specification [2]. The logic can be configured to perform burst transfers (TX and/or RX) of any specified size.
The master handler uses one link for write- and read-transfers, to reduce the complexity of the Internal Bus interconnect network, however, the handler considers the read access (TX data) having the highest priority. This means in fact that the TX FIFO is filled to a certain limit before the RX FIFO is emptied completely, however, it avoids TX FIFO underflows, and thus corrupted data to be transmitted to the network.

Information:

Note that the data stored in the RX FIFO is always copied to the memory completely, however, depending on the system's performance the last words (e.g. CRC32 of Ethernet frame) of a received frame might be copied after the transmission of a packet (e.g. auto-response). Nevertheless, this issue can be neglected since the delay is shorter than the reaction time of the host (e.g. interrupt latency).

The TX FIFO is filled to a certain level independent of the TX packet length, hence, the Internal Bus Memory Mapped Master reads data which is not part of the TX buffer. However, since the openMAC DMA requests for the correct data length, the unnecessary data patterns are dumped by the master handler by asserting an asynchronous reset signal connected to the TX FIFO. This avoids the transfer of incorrect TX data words in subsequent packet transfers.

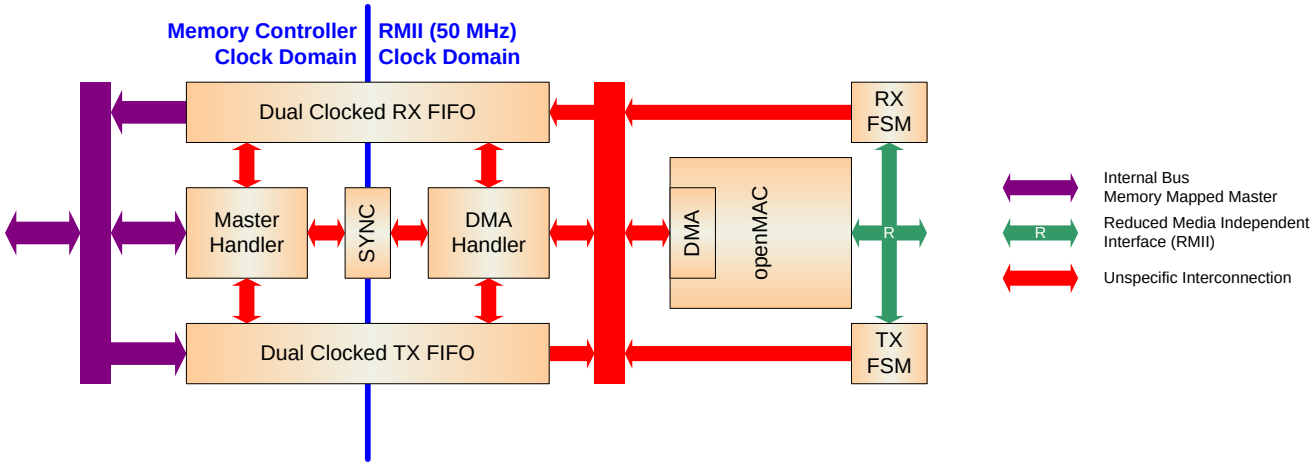

Fig. 8: Block diagram Internal Bus Memory Mapped Master

3.2.3.2 Performance Consideration

The ability to store TX and RX packets in external memory devices enables the system designer to reduce costs, since the FPGA size can be reduced. Nevertheless the choice of the used external memory type and the interface affects the performance of the DMA functionality of openMAC. Generally the following rules apply:

OpenMAC DMA performance rule:

- **Always use low-latency and deterministic memory technologies (SRAM) for TX packet storage!**
- **If you want to use dynamic memory (e.g. SDRAM), it is impossible to store TX packets externally. Otherwise the system will fail randomly by transmitting corrupted Ethernet packets.**
- **Always use SRAM with 10 ns speed grade and at least a data width of 16 bit.**

3.2.4 HUB (openHUB)

It is common to provide at least two Ethernet ports for industrial applications to allow flexible cabling of several nodes without any other equipment (e.g. extra hubs). The openHUB component integrates a low-latency RMII hub into the same FPGA, thus no extra IC is necessary! Since RMII is used the hub implementation requires very sparse resources (about 60 LEs and no M9Ks in Cyclone 4).
OpenHUB can be configured to a 3-port-hub (one port used for openMAC, the other two used for phys) or be equipped with more ports, however, the utilization increases accordingly.
During runtime the hub allows the following extra features:

- Enable/disable certain hub port:
  This ability is currently not used in the implementation, thus, every port is enabled.
- Provide port number of currently received frame:
  The port number is forwarded to the openMAC's receive descriptor and thus can be read back by the PCP.

3.2.5 Anti-Distortion-Filter (openFILTER)

Components for industrial environment must consider distortions and avoid errors. The POWERLINK IP-core is equipped with one Anti-Distortion-Filter per Ethernet phy to prevent distortion propagation to the POWERLINK node and across the whole network.
The openFILTER implementation requires very sparse FPGA resources (about 60 LEs and no M9Ks in Cyclone 4). The filter simply monitors the RX Data Valid line from the phy (Crsdv) and forwards the state as long as no condition is violated:

- Minimum packet length (8+64 bytes, 5.12µs)
- Maximum packet length (2048 bytes[10], 163.84µs)
- Minimum IPG (8 bytes, 0.64µs)

It has to be mentioned that the filter implementation does not use any buffering mechanism, thus there is no complete protection against distortions. However, openMAC does no interpretation of invalid packets too.

3.2.6 Phy Management (openMAC MII)

The Ethernet phys connected to the POWERLINK IP-core have to be configured for the certain application. This is done via dedicated configuration and status registers predefined by IEEE 802.3. These registers are stored physically in the Ethernet phys and are accessible via a serial management interface (SMI). The openMAC MII core enables the communication to the connected phy(s) via a predefined protocol.
Regularly all available Ethernet phys are interconnected to a bus on the PCB, but some evaluation board manufacturers connect the SMI of every phy to the FPGA separately. However, it is essential to set different hardware addresses to the phys, since only one openMAC MII core is used to communicate with several phys.
The openMAC MII component drives no interrupt line to the PCP therefore polling is necessary.

3.3 Process Data Interface

The Process Data Interface IP-core (PDI) includes the following features (refer to Fig. 9):

- PDO data exchange via triple buffer implementation (“3Buffer PDO Logic” and “1-to-3”)
- Simultaneous access from PCP and AP side
- Synchronizer considering arbitrary clock ratios between PCP and AP domain (and vice versa)
- Synchronization Interrupt Request Generator (“SYNC IRQ”)

The overview in Fig. 9 gives the structure of the PDI core. The Asynchronous TX/RX Buffers are implemented as simple buffers in the DPRAM. Thus the software design must consider data coherency in this memory regions.
The T/RPDO buffers are implemented as triple-virtual-buffers to avoid data access on the same content from different ports (PCP/AP) by hardware. The “3Buffer T/RPDO Logic” manages the virtual PDO buffer access from consumer and producer side (PCP or AP respectively). This logic is implemented in the PCP clock domain to achieve immediate virtual buffer switching (one cycle delay) on the PCP side of the PDI. A buffer switch-over command from the AP side must cross a synchronizer chain (two stages) to the PCP clock domain and vice versa. Thus in total 3 PCP clock cycles + 2 AP clock cycles are required to initiate a virtual buffer change.
The Control/Status Register is not mapped to the DPRAM (except some PCP/AP shared registers), since the PCP and AP must access different functions (e.g. PCP triggers IRQ and AP does acknowledgment).

[10] Ethernet maximum packet length is set to 8+1518 bytes, however the set limit allows to reduce HW complexity.

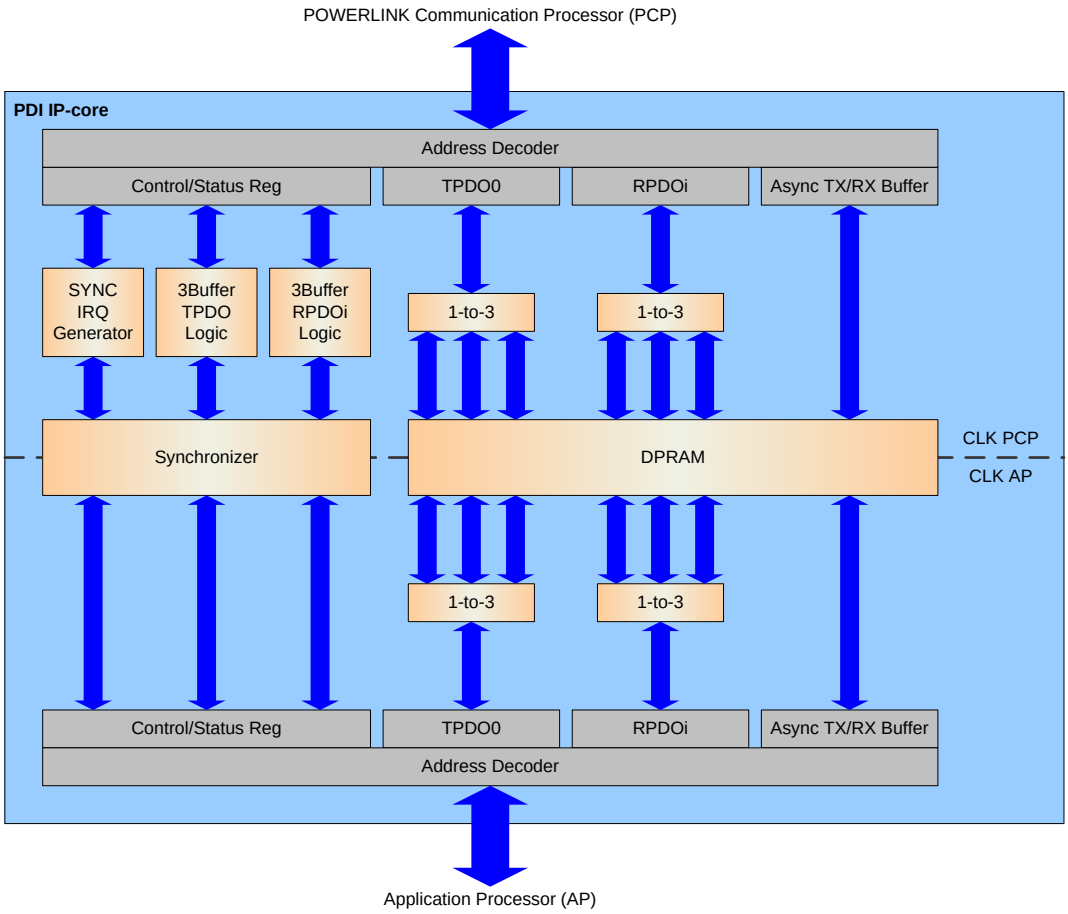

Fig. 9: PDI IP-core architecture overview

3.3.1 SYNC IRQ Generator

In order to synchronize the AP to the POWERLINK network there are several approaches possible depending on the application.
In general the synchronization to the SoC achieves the lowest possible jitter (equal to the SoC jitter on the network). If the synchronization to the SoC is not required, other packets (e.g. SoA) can be used for the AP SYNC IRQ.

The SYNC IRQ Generator allows two modes to synchronize the AP:

- MAC timer:
  The MAC timer (free-running 32bit counter, driven with 50MHz) is compared to the CMP_VAL_TOG[11]. When the values match, a signal is toggled. The toggling signal is transferred to the AP's clock domain and triggers the AP IRQ.
- Software:
  If the application does not require synchronizing with very low jitter, the PCP can trigger the SYNC IRQ by software.

Fig. 10 shows the structure of the SYNC IRQ generation, which is implemented in the AP clock domain. The PCP signals are synchronized via a 2-stage synchronizer chain (two FF) to avoid metastable circuits. Depending on the mode signal – which is set in the Control/Status register in the PCP – the IRQ is asserted by the "Set" signal (software IRQ).
The other possibility – in order to generate a low-jitter SYNC IRQ to the AP – is to use the InIrq. InIrq is connected to the toggle port of the MAC Timer Compare unit and triggers the OutIrq signal.

[11] Refer to 4.2.1

Independently of the selected mode the AP should acknowledge an asserted IRQ by writing to the Control/Status register. Otherwise the FSM gets stuck in the “wait4ack” state – refer to the FSM in Fig. 11.

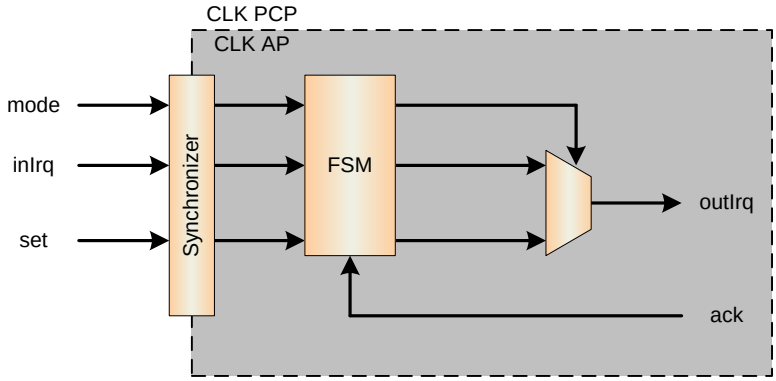

Fig. 10: SYNC IRQ Generator

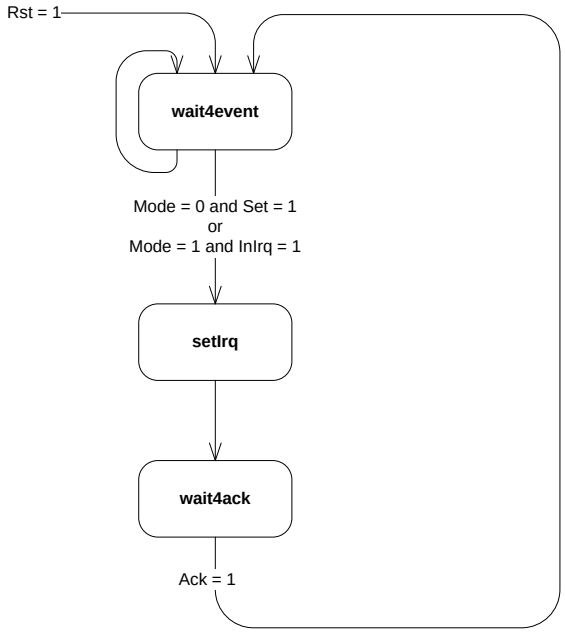

Fig. 11: SYNC IRQ Generator - FSM

3.3.2 Synchronizer

The synchronizers used in the PDI are built out of 2-stage flip flop chains. This approach is valid for signals, which smoothly change their states (refer to Fig. 12). In order to transfer a pulsing signal further considerations are required:
The pulse signal is asserted during the source clock cycle. When the destination clock rate is faster a simple 2-stage synchronizer is possible. If the destination clock cycle is longer than the source, the synchronizer will not recognize the pulse confidently. Thus the pulse has to be converted into a level changing signal (toggle). The toggling signal is transferred into the destination domain with a 2-stage synchronizer. Afterwards an edge detector decodes the level change into a pulse – for the pulse transfer the any output has to be used – valid for the destination clock domain.

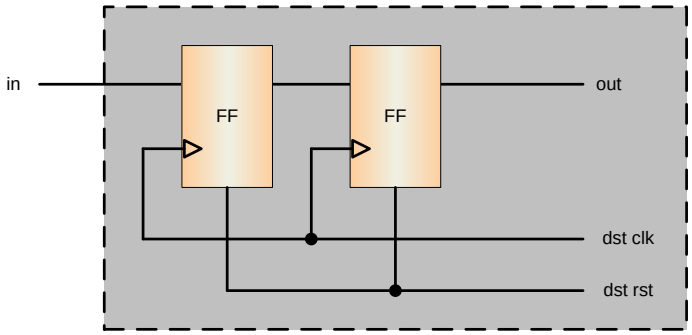

Fig. 12: Simple 2-stage Synchronizer

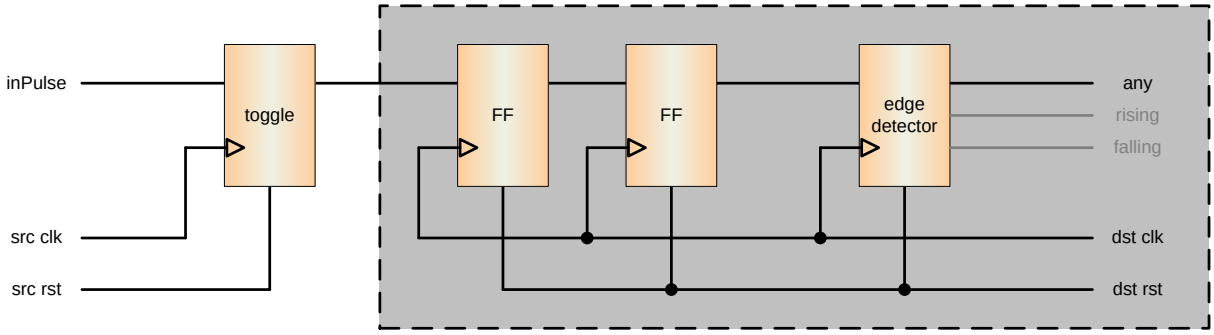

Fig. 13: 2-stage Synchronizer for pulse transfer

3.3.3 Triple Buffer Logic

The triple buffer logic is used to translate the input addresses (from the PCP respectively AP) to output addresses used for the DPRAM. The DPRAM stores three virtual buffers, which are accessible from consumer and producer side simultaneously. However, the consumer may not read from a virtual buffer that is written by the producer. Thus the Triple Buffer Logic implements a locking mechanism to avoid access to the same virtual buffer.
In order to change the virtual buffer the producer/consumer triggers a buffer change, what frees or validates a buffer and switches over to an unused one. Since the producer and consumer interface of the Triple Buffer Logic may be located in different clock domains, the Triple Buffer Logic is located in the PCP's clock domain. Please consider that the PCP acts as producer for RPDO and as consumer for TPDO.
Fig. 14 shows the System Overview of the Triple Buffer Mechanism. The system is divided into two present clock domains. The output address (for PCP and AP) is calculated with an adder using the input address as the first addend and the offset of the virtual buffer (constant) in the DPRAM as the second addend, which is selected by the Triple Buffer Logic.
As already mentioned the Triple Buffer Mechanism is implemented in the PCP's clock domain, therefore the trigger signal asserted by the AP and the select virtual buffer signal (SelVBuf) asserted by the PCP are synchronized to the respective destination clock domain. The select virtual buffer signals are used in the Control/Status register to read out the current selected virtual buffer – for debugging purpose only.

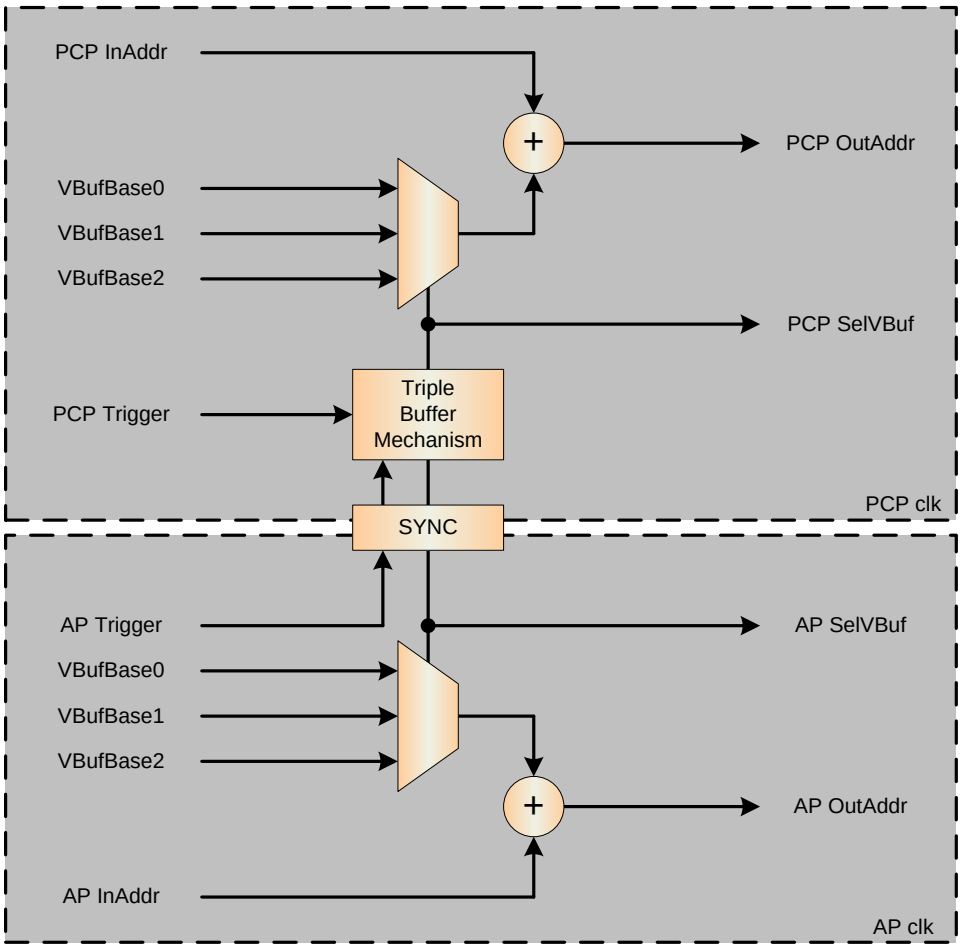

Fig. 14: Triple Buffer Logic System Overview

Variable[12] definition

The Triple Buffer Mechanism uses two variables to select the next virtual buffer for the consumer/producer.

- VALID:
  Is set by the producer and flags the most current data to the producer.
- LOCKED:
  Is set by the consumer and tags the virtual buffer that is used by the consumer.
- CURRENT:
  Is set by the producer and identifies the currently used virtual buffer by the producer.

Producer switch

When the producer sets the trigger signal, the Triple Buffer Mechanism has to decide which buffer has to be selected next.
Assume the example in Tab. 4 to understand the decision reached by the producer:
Buffer zero is locked by the consumer for data processing purpose and buffer one is validated by the producer. Thus the “next free virtual buffer” can be assigned to the CURRENT variable.

Tab. 4: Example – Decision LUT for "next free virtual buffer"

Variable	VBuf2	VBuf1	VBuf0
LOCKED	0	0	1

[12] The expression “VARIABLE” used in this context is not equal to a variable used in VHDL.

VALID	0	1	0
CURRENT	1	0	0

Out of the LUT (Tab. 4) the logic expression can be derived:

```
CURRENT <= NOT LOCKED AND NOT VALID
```

Consumer switch

When the consumer sets the trigger signal, the Triple Buffer Mechanism has to decide which buffer has to be selected next.
For the consumer case it simply switches to the valid virtual buffer. In detail the consumer assigns the LOCKED to the VALID variable.
If the consumer triggers a buffer change and the producer has not yet validated a new buffer, the consumer will stay at the currently locked buffer, since there is no “newer” data available in the moment.

Consider special case

After the consumer has switched to the most current data – LOCKED equals VALID – the logic expression derived from Tab. 4 would fail. Thus the VHDL implementation uses a process with a VALID variable[13] before CURRENT is assigned.
This design approach furthermore overcomes the case of buffer switch from consumer and producer at the same time. The consumer will receive the just validated virtual buffer from the producer. Due to completeness it has to be mentioned that the producer will switch according to the example in Tab. 4.

3.3.4 Time Synchronization

In order to synchronize the POWERLINK node to the network an optional synchronization feature is added to the PDI. The component provides the following information to the AP:

- RELATIVE_TIME (source: SoC packet)
- NETTIME (source: SoC packet)
- TIME_AFTER_SYNC (generated by hardware counter)

The RELATIVE_TIME and the NETTIME is sourced by the latest SoC packet received, which is set by the MN. The patterns have a size of 64 bits and are handled with a double buffering system, hence the AP reads the data correctly within the SYNC interrupt context. The double buffers are switched by the SYNC interrupt assertion.

Information:

If the SYNC interrupt is not confirmed by the AP before the subsequent interrupt assertion, the double buffer is not switched! Hence the AP has to react within the SYNC interrupt period!

The TIME_AFTER_SYNC pattern refers to a 16 bit counter which is reset by the SYNC interrupt edge. The AP has to read out that register, which refers to the time between the assertion of the interrupt and the read-out event. This delay is necessary for correct node synchronization. The time base for that timer value is determined by the AP clock rate (e.g. 50 MHz refers to 20 ns per tick).

For further information refer to [3].

3.4 Asynchronous 8/16bit Parallel Interface

In order to provide high performance access to the PDI from external MCUs the AP-side of the PDI is connected to the 8/16bit Parallel Interface IP. This parallel interface is asynchronous and uses common signals provided by many MCUs (with external bus feature) on the market.

[13] The expression “variable” used in this context is equal to the variable usage in VHDL.

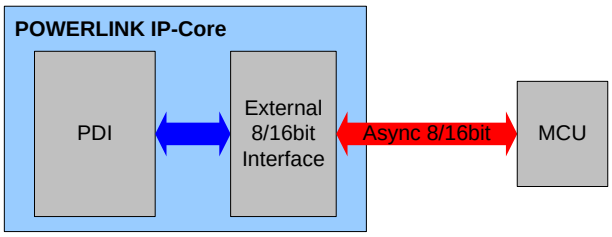

Fig. 15: Asynchronous 8/16bit Parallel Interface Overview

The external interface is mapped to the Internal Bus Interface signals of the PDI. Thus from the MCU's point of view the PDI works like a SRAM component with a predefined timing specification.

Features

The Asynchronous 8/16bit Parallel Interface supports several features as follows:

- Asynchronous 8 or 16bit address-/data-bus interface (SRAM-like)
- Low- or high-active control signals (e.g. CS)
- Little or Big Endian

The configuration is done via generics!

3.4.1 General Description

The Asynchronous 8/16bit Parallel Interface allows to connect external MCUs to the AP side of the PDI by converting the external bus to the Internal Bus signals. The IP converts the 8 or 16bit data width to 32bit by generating the appropriate address-, byte enable- and data signals.
The IP-core has synchronizer chains included for all input signals. The input data is captured by register that uses the WR signal as clock. The register value is transferred to the AP clock domain (by synchronizer) and assigned to the writedata signal vector depending on the address (byteenable and address). The write signal is asserted (pulse) by an edge detector, which is sensitive to the falling write edge. The optional ACK signal is assigned to the output-enable signal for the tri-state buffer or to the write strobe.

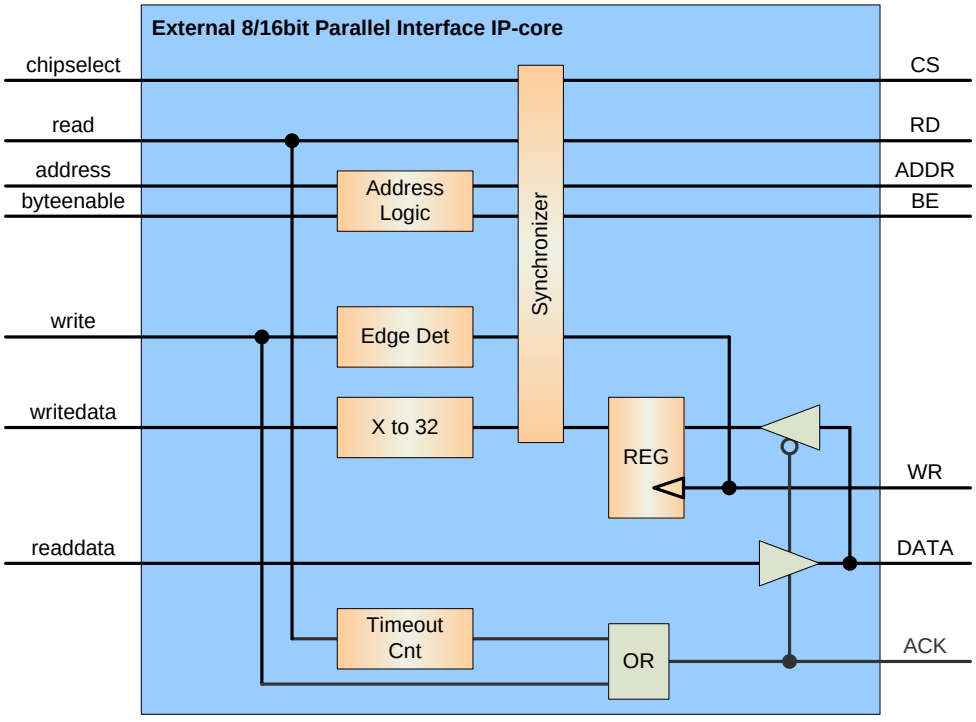

Fig. 16: Asynchronous 8/16bit Parallel Interface IP-core

Information:

The READY signal is optional, since the interface's timing is fixed after synthesis.

3.4.2 Timing Specification

Tab. 5: Timing Specification of Asynchronous 8/16bit MCU Interface

Symbol	Value		Comment
	Min	Max	
t_{SC}	0ns	-	The CS signal has to be asserted before or with write or read signals.
t_{HC}	0ns	-	The CS signal has to be deasserted after or with write or read signals.
t_{PWR}	20ns	-	Write pulse Note: The WR signal can be hold for a certain time to insert wait-states.
t_{SD}	5 ns	-	Data Setup to Write End
t_{HD}	2 ns	-	Data Hold from Write End
t_{AA}	60ns	80ns	Address access time after the data bus is driven with valid data.
t_{OHA}	20ns	40ns	Valid output data to be held on the data bus (before changing to HIGH Z).
t_{WAD}	40ns	60ns	ACK signal assertion after Write END
t_{ARD}	20ns	-	ACK signal assertion duration for read
t_{AWR}	20ns	20ns	ACK signal assertion duration for write
t_{IWR}	20ns	-	Idle time after write access before subsequent write or read
t_{IRD}	40ns	-	Idle time after read access before subsequent write or read

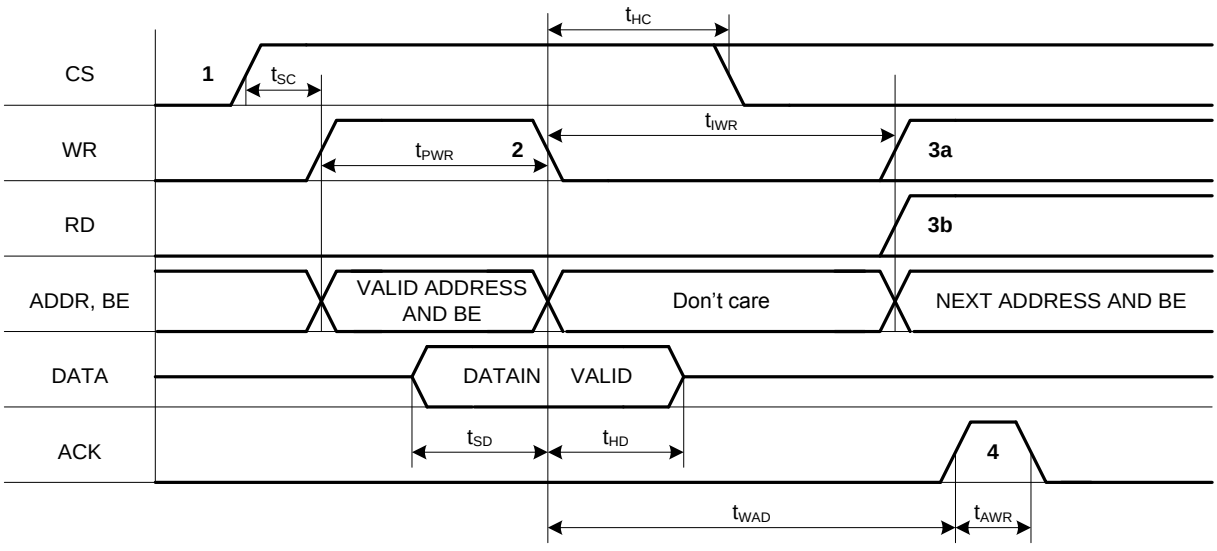

Fig. 17: Write Access Timing

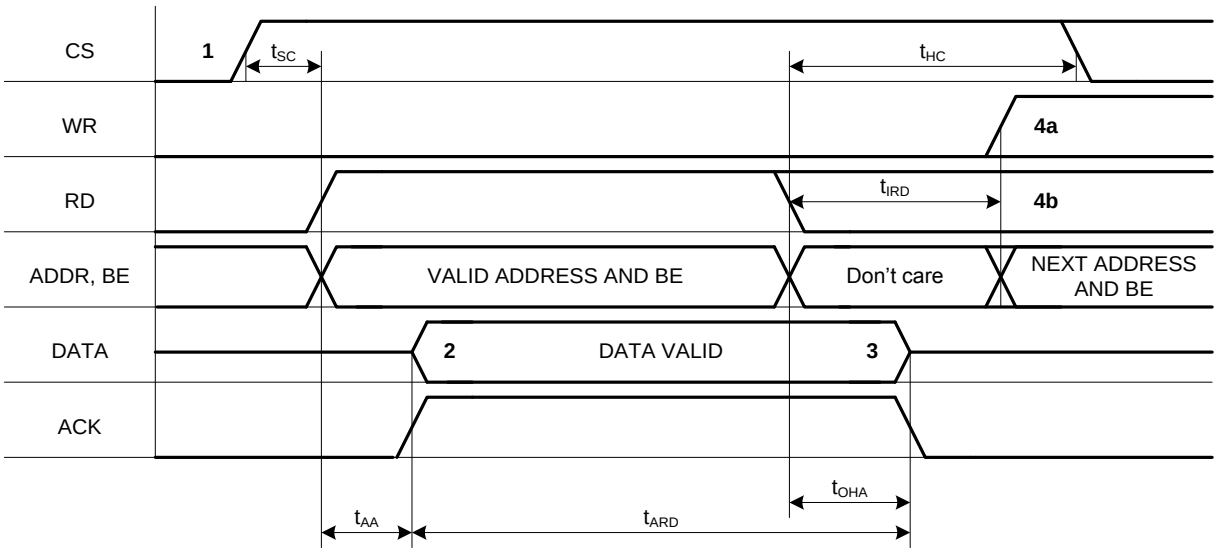

Fig. 18: Read Access Timing

3.5 SPI

The PDI is accessible via a common serial interface – SPI. The POWERLINK IP-core acts as a SPI slave by providing the content of the PDI's memory content.
The SPI slave is able to receive/transmit 8 bit data frames. The SPI frames are used to address the PDI and transfer data.

Features

The SPI PDI core implements the following features:

- Slave only
- 8bit data
- Support of all four modes (CPOL = 0 or 1 and CPHA = 0 or 1)
- Shift direction: MSB first only

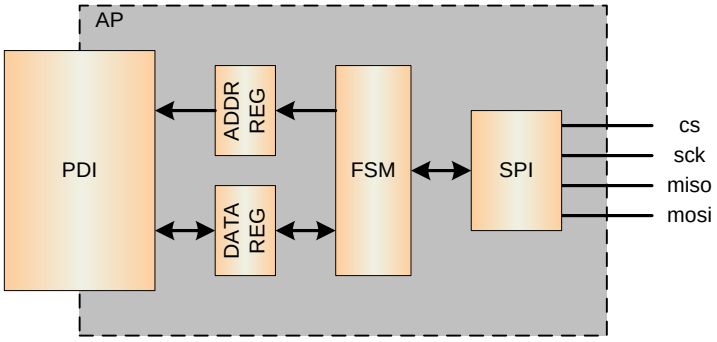

Fig. 19: SPI IP-core architecture overview

Data Transmission

SPI is build up with a Master and Slave shift register in a ring connection. The SPI Master drives a clock signal that controls shifting and capturing of the SPI devices. Depending on the mode the data is captured respectively shifted at the rising or falling edge. The four different modes are selected with CPOL and CPHA.
The data transmission is armed by the SPI Master, which asserts the SPI Select (SS) signal. This triggers the MISO and MOSI data lines to exit the High 'Z' level and enter logical states. The SPI clock (SCK) con-

trols the shift registers and capturing of the SPI Master and Slave. After a frame is transferred the SPI Master deasserts the SPI Select (SS) signal or initiates another SPI transfer.

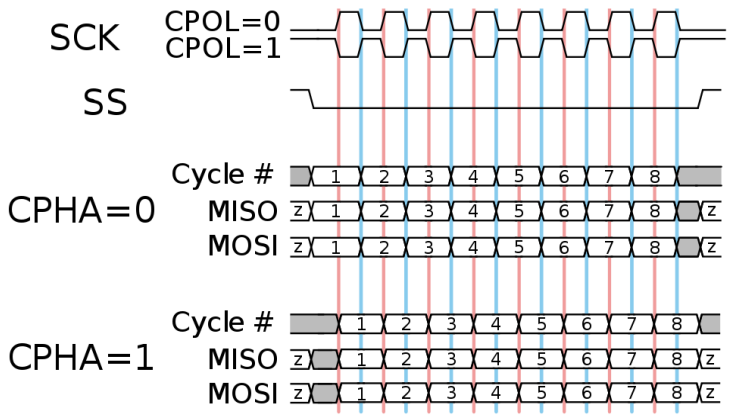

Fig. 20: SPI timing [1]

3.5.1 Communication Protocol

The communication via SPI requires coding the address and data into single SPI frames. Due to the high address range of the PDI the SPI core stores the address (ADDR(14..0)) and reuse it as long as they are unchanged. In case of single write/read transfers the address bits 4 to 0 are transmitted before the final data transfer.
In order to reduce the overhead a sequence of write/read transfers can be initiated. Independent of the write/read command the address register is incremented after each write/read!

The protocol differs between two types of frames:

- Command Frame
- Data Frame

Command Frame (Master to Slave)

The command frame controls the SPI FSM and sets e.g. the higher address register.
The general structure is shown in the following. For the CMD CODE meaning refer to Tab. 6.

CMD Frame							
7	6	5	4	3	2	1	0
CMD(2..0)			PAYLOAD(4..0)				

Tab. 6: Command Frame Codes

CMD	CMD NAME	PAYLOAD	Description
0b100	HIGHADDR	ADDR(14..10)	Sets the higher five address bits to the address register.
0b101	MIDADDR	ADDR(9..5)	Sets the middle five address bits to the address register.
0b110	WR	ADDR(4..0)	Sets the lower five address bits to the address register. The following frame must include the data to be written!
0b111	RD	ADDR(4..0)	Sets the lower five address bits to the address register. The following frame includes the read data!
0b001	WRSQ	BYTES	Initiates a write sequence of BYTES+1. The address register is incremented by one. Note: BYTES represents values 0 to 31
0b010	RDSQ	BYTES	Initiates a read sequence of BYTES+1. The address register is incremented by one. Note: BYTES represents values 0 to 31
0b011	LOWADDR	ADDR(4..0)	Sets the lower five address bits to the address register.

0b000	IDLE	X	Idle command (FSM does no interpretation)

Data Frame

After a WR (0b110) or RD (0b111) command frame the following frame must be a data frame with the following structure. After the transfer of a data frame the address register is incremented by 1 byte!

DATA Frame							
7	6	5	4	3	2	1	0
DATA(7..0)							

Full-Duplex

SPI supports full duplex data transfer, since the master and slave shift in and out data simultaneously. However the SPI slave IP-core does not use this feature and simply shifts out old data if there is no transfer to the master required. Depending on the protocol the master must interpret the content as data or not.

Data Flow

After reset the address register (ADDR REG) of the SPI core is set to zero. Depending on the write or read address the SPI master must set the higher address bits to the required value.
The following examples show some practicable scenarios. In Fig. 21, Fig. 22 and Fig. 23 the data flow is visualized.
In order to avoid double-addressing[14] the software driver should store and consider an address shadow register!

Examples:

The SPI core ADDR REG is set to 0x0000. The SPI master wants to write data to 0x0004:
- **CMD Frame "WR" with ADDR(4..0) = 0b00100**
- **Data Frame from SPI master**

The SPI core ADDR REG is set to 0x0000. The SPI master wants to write data to 0x2800:
- **CMD Frame "HIGHADDR" with ADDR(14..10) = 0b01010**
- **CMD Frame "WR" with ADDR(4..0) = 0b00000**
- **Data Frame from SPI master**

The SPI core ADDR REG is set to 0x3000. The SPI master wants to read data from 0x308F:
- **CMD Frame "MIDADDR" with ADDR(9..5) = 0b00100**
- **CMD Frame "RD" with ADDR(4..0) = 0b01111**
- **Data Frame from SPI slave**

The SPI core ADDR REG is set to 0x308F. The SPI master wants to read data from 0x3090:
- **CMD Frame "RD" with ADDR(4..0) = 0b10000**
- **Data Frame from SPI slave**

The SPI core ADDR REG is set to 0x100F. The SPI master wants to write 10 bytes starting at 0x1000:
- **CMD Frame "LOWADDR" with ADDR(4..0) = 0b00000**
- **CMD Frame "WRSQ" with BYTES = (10-1)**
- **1st Data Frame from SPI master**

[14] Double-addressing means in this context to write to the slave parts of the address twice.

- **2nd Data Frame from SPI master**
- **…**
- **10th Data Frame from SPI master**

The SPI core ADDR REQ is set to 0x200C. The SPI master wants to read 32 bytes starting at 0x2000:

- **CMD Frame "LOWADDR" with ADDR(4..0) = 0b00000**
- **CMD Frame "RDSQ" with BYTES = (32-1)**
- **1st Data Frame from SPI slave**
- **2nd Data Frame from SPI slave**
- **…**
- **32nd Data Frame from SPI slave**

The SPI core ADDR REQ is set to 0x300F. This SPI master wants to write 48 bytes starting at 0x3000:

- **CMD Frame "LOWADDR" with ADDR(4..0) = 0b00000**
- **CMD Frame "WRSQ" with BYTES = (32-1)**
- **1st Data Frame from SPI master**
- **2nd Data Frame from SPI master**
- **…**
- **32nd Data Frame from SPI master**
- **CMD Frame "WRSQ" with BYTES = (16-1)**
- **33rd Data Frame from SPI master**
- **34th Data Frame from SPI master**
- **…**
- **48th Data Frame from SPI master**

Information:

In case of a read command (SPI Master reads from SPI Slave - Fig. 23) the SPI Master must send an IDLE command (CMD = 0b000) in order to not confuse the SPI PDI FSM!

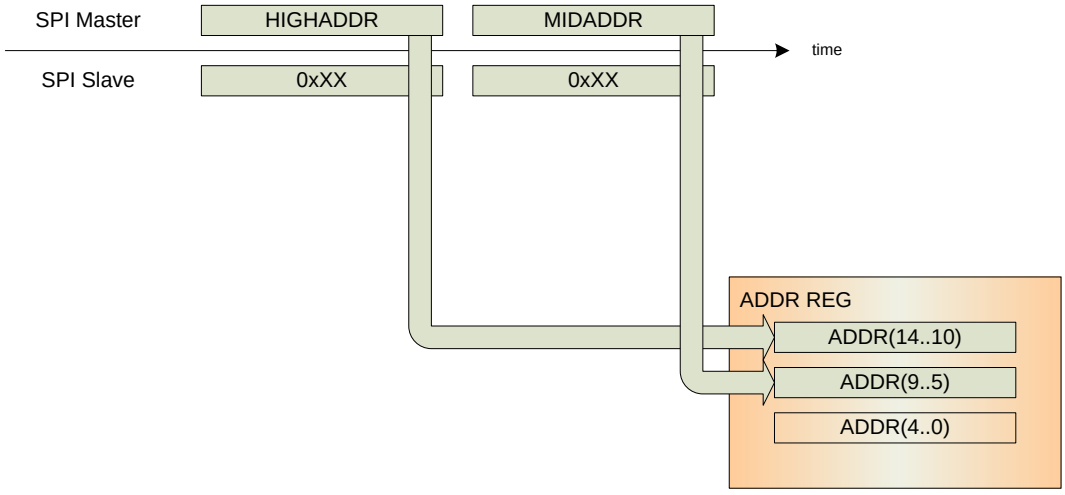

Fig. 21: SPI Data Flow – Addressing

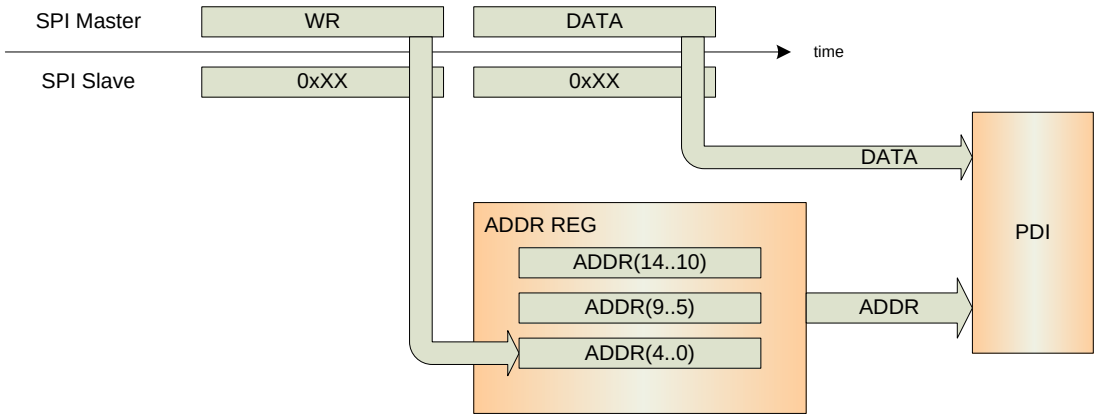

Fig. 22: SPI Data Flow – Write Data

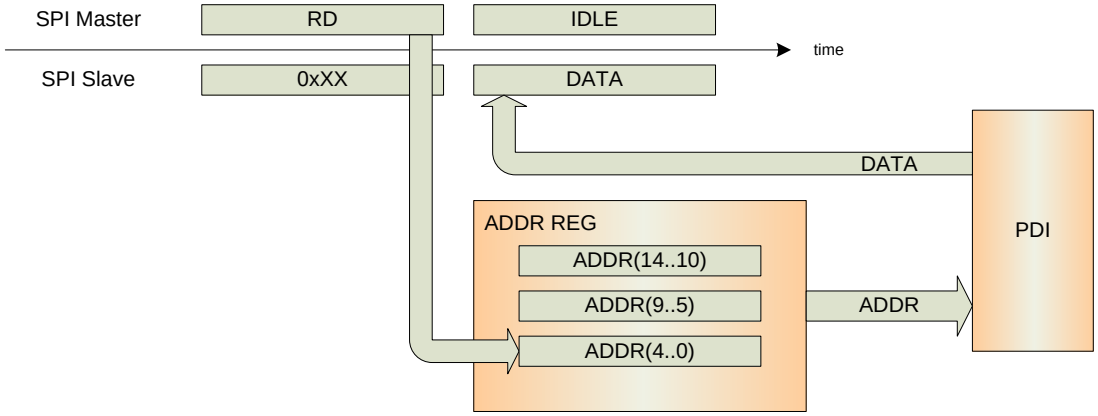

Fig. 23: SPI Data Flow – Read Data

3.5.2 Finite State Machine (FSM)

The FSM controls the PDI access depending on the SPI data protocol (refer to 3.5.1). Furthermore a reset sequence is necessary to successfully wake up the SPI PDI. After reset the FSM stays in the reset state until a dedicated SPI frame is received.
In the decode state the command of the SPI frame is interpreted.
Assume a “WRSQ” command which enters the waitwr state next. In the transition the variable writes is set to the BYTE+1 pattern. The variable writes is decremented after every write and verified to repeat the sequence (writes /= 0). In case of a simple “WR” or “RD” command the variable writes/reads respectively are set to one.

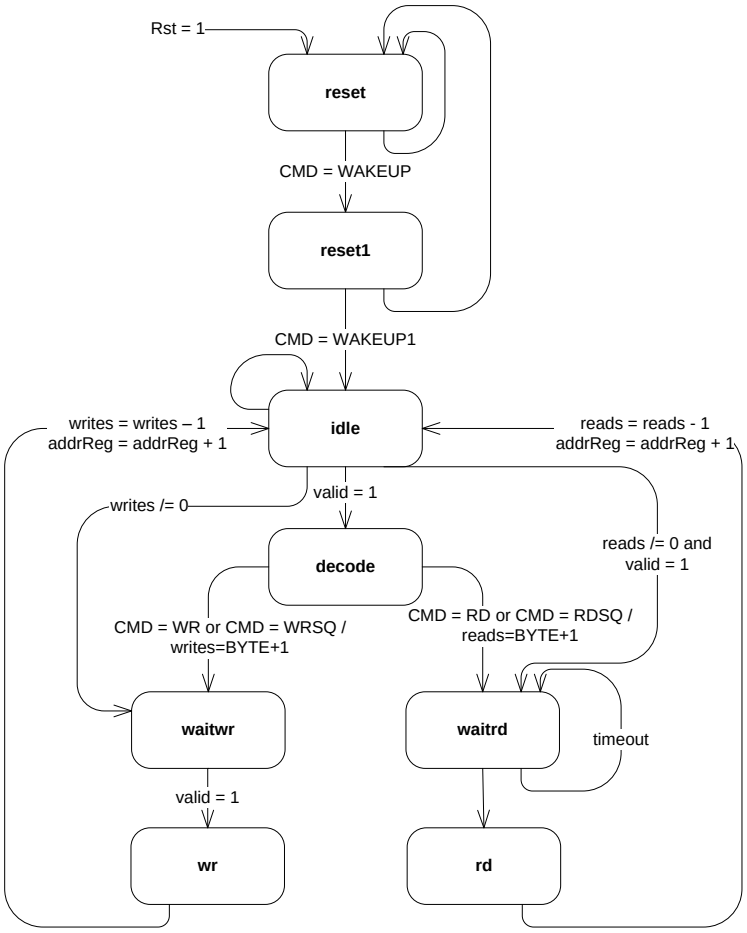

Fig. 24: SPI FSM

3.5.3 Wake Up

The wake up functionality enables a defined boot sequence of the SPI slave. After hardware reset the PDI SPI is in reset state (refer to Fig. 24) and waits for the wake up frame transmitted by the SPI master. If the received frame is correctly decoded as WAKEUP frame, the PDI SPI is ready to use by entering the idle state.

Wake Up Frame

The WAKEUP and WAKEUP1 frames are only valid in the reset state. If these frames are received in the idle state, the FSM will interpret these commands as idle (since the highest three bits are zero).

WAKEUP Frame							
7	6	5	4	3	2	1	0
0b000			0	0x3			

WAKEUP1 Frame							
7	6	5	4	3	2	1	0
0b000			0	0xA			

3.6 I/O Port

TBD

4 Interface Definition

The Interface Definition is required to access different functionalities of the POWERLINK IP-core. This information is useful for software designer to develop device drivers.

4.1 POWERLINK

Depending on the configuration the POWERLINK IP-core consist of different Internal Bus Memory Mapped Slave connections. Refer to Tab. 7 and its footnotes.

Tab. 7: Available Memory Mapped Interfaces

Nr	Configuration	openMAC[15]			PDI		PORTIO
	Memory Mapped Connection Points	CMP	REG	BUF	PCP	AP	SMP
1	Simple I/O (no AP)	✓	✓	✓	×	×	✓
2	Internal AP	✓	✓	✓	✓	✓	×
3	External AP (e.g. MCU or DSP) for high data throughput	✓	✓	✓	✓	✓[16]	×
4	External AP (e.g. MCU or DSP) for low data throughput	✓	✓	✓	✓	✓[17]	×

The memory mapping of the mentioned Connection Points in Tab. 7 are described in detail on the following pages.

In order to share a detailed description two examples are assumed in the following:

Example 1

An external AP is used (e.g. ARM MCU) connected to the POWERLINK IP-core via SPI – refer to Tab. 7 / 4 – to reduce PCB size. The POWERLINK component will include the openMAC interface (mandatory) and the PDI connection points. The PDI PCP interface is connected to the Internal Bus. The PDI AP interface uses the SPI protocol for data exchange to external components.

Example 2

An external AP is again available and should be connected via a 16bit parallel interface to achieve a higher data throughput. In Tab. 7 the 3rd entry has to be considered: Once again the mandatory openMAC interfaces and the PCP connection point to the PDI are generated as Internal Bus Memory Mapped Slave Interfaces. The AP port to the PDI uses a generic parallel interface protocol that is compatible to most of the MCU available on the market.

4.2 OpenMAC

The openMAC IP is assembled to manage the Ethernet interfacing and POWERLINK specific hardware acceleration. Beside general configuration and status registers the MAC uses TX and RX descriptor blocks and RX filters. This functionality is accessible via the REG connection (4.2.2).
In order to achieve very low jitter synchronization a timer-triggered interrupt must be generated what is realized with the MAC-timer and an additional compare unit accessible via CMP (4.2.1).
The packet buffer is realized with a DPRAM connected to the MAC and the second port is accessible via the “BUF” interface (4.2.2.2).

[15] OpenMAC is mandatory for every configuration.
[16] For high data throughput to FPGA-external components it is recommended to use the parallel 8/16bit interface configuration.
[17] For low data throughput it is sufficient to use SPI.

4.2.1 MAC Timer Compare Register (CMP)

The CMP interface provides access to the MAC-timer compare unit to generate MAC-timer-triggered interrupts. The compare unit is controlled via the CMP_CNTR and CMP_CNTR_TOG registers by write accesses.
Status and the current MAC time can be read out of the CMP interface.
Furthermore the CMP unit outputs a toggling signal that changes its level at a second compare value match to the MAC-time. This toggling signal can be used for the AP synchronization.

In order to acknowledge the CMP_IRQ a write access to CMP_VAL_IRQ or CMP_VAL_TOG has to be performed.

CMP write access

CMP (WR)	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x000C	-								-								-								CMP_CNTR_TOG							
0x0008	CMP_VAL_TOG																															
0x0004	-								-								-								CMP_CNTR							
0x0000	CMP_VAL_IRQ																															

0x0004	CMP_CNTR						
7	6	5	4	3	2	1	0
-							EN_IRQ

Bit	Name	Default	Description
0	EN_IRQ	0	0 = CMP IRQ is disabled, 1 = CMP IRQ is enabled

0x000C	CMP_CNTR_TOG						
7	6	5	4	3	2	1	0
-							EN_ TOG

Bit	Name	Default	Description
0	EN_TOG	0	0 = TOG IRQ is disabled, 1 = TOG IRQ is enabled

CMP read access

CMP (RD)	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x000C	-								-								-								CMP_STAT_TOG							
0x0008	CMP_VAL_TOG																															
0x0004	-								-								-								CMP_STAT							
0x0000	MAC_TIME																															

0x0004	CMP_STAT						
7	6	5	4	3	2	1	0
-						IRQ	EN_IRQ

Bit	Name	Default	Description
0	EN_IRQ	0	0 = CMP is disabled, 1 = CMP is enabled
1	IRQ	0	0 = No IRQ, 1 = IRQ was generated due to MAC_TIME = CMP_VAL_IRQ

0x000C CMP_STAT

7	6	5	4	3	2	1	0
-						TOG	EN_TOG

Bit	Name	Default	Description
0	EN_TOG	0	0 = CMP TOG is disabled, 1 = CMP TOG is enabled
1	TOG	0	Represents the level of TOG

4.2.2 MAC Register (REG)

The MAC register includes several registers and components as mentioned in Tab. 8.

Tab. 8: MAC offset mapping

Offset	Component
0x0000	openMAC Control/Status Register
0x0800	openMAC filter DPRAM
0x0C00	openMAC descriptor DPRAM
0x1000	openMAC Phy management (MII)
0x1010	openMAC IRQ table
0x1020	openMAC DMA Observer

4.2.2.1 openMAC IRQ table

MAC_REG_BASE + 0x1010 openMAC IRQ table

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-														RX	TX

Bit	Name	Default	Description
0	TX	0	This bit represents a pending TX interrupt (1 = pending IRQ).
1	RX	0	This bit represents a pending RX interrupt (1 = pending IRQ).

4.2.2.2 openMAC DMA Observer

MAC_REG_BASE + 0x1020 openMAC IRQ table

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-							RD	-							WR

Bit	Name	Default	Description
0	WR	0	This bit is asserted if a DMA write error occurs. Note: This bit holds the error state until a hardware reset is performed.
8	RD	0	This bit is asserted if a DMA read error occurs. Note: This bit holds the error state until a hardware reset is performed.

4.2.3 MAC Buffer (BUF)

The MAC Buffer interface “BUF” is directly mapped to the packet buffer DPRAM. In order to find the base addresses of a packet the appropriate packet descriptor (including base address and further information) must be read first.

4.3 Process Data Interface (PDI PCP/AP)

The Process Data Interface (PDI) provides the interface between the PCP and AP, and is defined as follows. Fig. 25 shows the memory mapping of the Control/Status Registers and the Asynchronous and Process Data buffers.

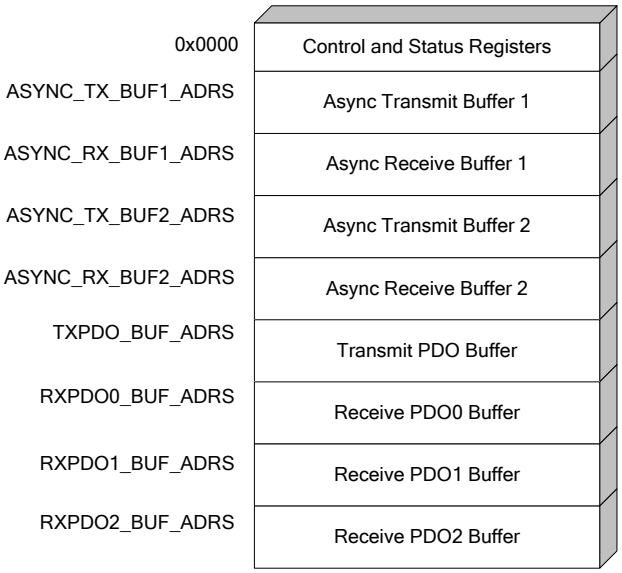

Fig. 25: Memory Map

4.3.1 Control and Status Register

The following table shows the memory map of the control/status register. Note that the implementation at hardware level is described only, hence, functionality provided by software are omitted.

Control Registers 0x0000																															
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x0000	MAGIC (RO)																															
0x0004	reserved																FPGA_REV (RO)														
0x0008 … 0x003C	DPRAM																															
0x0040 … 0x004C	2x DPRAM TIME_SYNC																															
0x0050	reserved																TIME_AFTER_SYNC														
0x0054	EVENT_ACK																ASYNC_IRQ_CTRL														
0x0058	TXPDO_BUF_ADRS (RO)																TXPDO_BUF_SIZE (RO)														
0x005C	RXPDO0_BUF_ADRS (RO)																RXPDO0_BUF_SIZE (RO)														
0x0060	RXPDO1_BUF_ADRS (RO)																RXPDO1_BUF_SIZE (RO)														

0x0064	RXPDO2_BUF_ADRS (RO)	RXPDO2_BUF_SIZE (RO)
0x0068	ASYNC_TX_BUF1_ADRS (RO)	ASYNC_TX_BUF1_SIZE (RO)
0x006C	ASYNC_RX_BUF1_ADRS (RO)	ASYNC_RX_BUF1_SIZE (RO)
0x0070	ASYNC_TX_BUF2_ADRS (RO)	ASYNC_TX_ BUF2_SIZE (RO)
0x0074	ASYNC_RX_BUF2_ADRS (RO)	ASYNC_RX_ BUF2_SIZE (RO)
0x0078	reserved	reserved
0x007C	reserved	reserved
0x0080	RXPDO0_ACK	TXPDO_ACK
0x0084	RXPDO2_ACK	RXPDO1_ACK
0x0088	reserved	SYNC_IRQ_CTRL (PCP only)
0x008C	reserved	
0x0090	reserved	
0x0094	LED_CNFG	LED_CTRL

4.3.1.1 Magic Number Register (MAGIC)

0x0000	MAGIC (RO)																													
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x50435000 = “PCP\0”																															

This register contains a magic number which is used to identify a valid DPRAM memory block.

4.3.1.2 FPGA Revision Register (FPGA_REV)

0x00004	FPGA_REV (RO)														
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Revision Nr. of FPGA Configuration															

This register shall be used by the AP software in order to recognize an FPGA configuration which does not match the current software.

4.3.1.3 Embedded Memory Block (DPRAM)

0x0008 … 0x003C	DPRAM																													
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DPRAM																															

This register is mapped to FPGA memory, hence, the content is defined by the software implementation.

4.3.1.4 Double-Buffered Embedded Memory Block for Time Synchronization (2x DPRAM TIME_SYNC)

0x0040 … 0x004C	2x DPRAM TIME_SYNC																													
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
2x DPRAM																															

This register is mapped to FPGA memory, hence, the content is defined by the software implementation. In addition a double-buffer mechanism is implemented, which is controlled by the SYNC IRQ routed to the AP. This realisation avoids invalid or corrupted data being read by the AP.

4.3.1.5 Time After Synchronization Interrupt (TIME_AFTER_SYNC)

0x00050								TIME_AFTER_SYNC (RO)							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TIME_AFTER_SYNC															

The TIME_AFTER_SYNC field provides a 16 bit clock tick value elapsed since the SYNC_IRQ. The clock ticks refer to the AP's clock (e.g. 50 MHz leads to 20 ns clock ticks). Note that the value starts counting when SYNC_INT is asserted and finally saturates at 0xFFFF, hence, the AP has to read out the value as soon as possible and only once!

4.3.1.6 Control Register of Asynchronous IR Signal (ASYNC_IRQ_CTRL)

0x0054								ASYNC_IRQ_CTRL (AP only)							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EN	-														

The asynchronous IRQ control is only accessible by the AP.

Bit	Name	Default	Description
15	EN	0	1 = Interrupt Mode, 0 = Polling Mode (IR signal deactivated)

4.3.1.7 Event Acknowledge (EVENT_ACK)

0x0056								EVENT_ACK							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	PL1	PL0	-	-	-	-	-	GE

The EVENT_ACK register can be accessed by the PCP and the AP respectively. The PCP sets an event by writing a '1'. The AP acknowledges the event by writing a '1' to the respective bit. The asynchronous interrupt is asserted if at least one of the event bits is set to '1'.
Example: In order to acknowledge the Event PL1 and SW, the AP has to write the pattern 0x0081.

Bit	Name	Default	Description
0	GE	0	Generic event set by PCP '1' = generic event occurred, '0' = no event Note: The AP has to read the registers EVENT_TYPE and EVENT_CODE, before sending an acknowledge!
6	PL0	0	Phy 0 link down '1' = Phy 0 link down event occurred, '0' = no event
7	PL1	0	Phy 1 link down '1' = Phy 1 link down event occurred, '0' = no event

4.3.1.8 Transmit PDO Message Buffer Size Register (TXPDO_ BUF_SIZE)

0x0058								TXPDO_BUF_SIZE (RO)							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SIZE															

The size of the transmit PDO message buffer in Bytes.

4.3.1.9 Transmit PDO Message Buffer Address Register (TXPDO_ BUF_ADRS)

0x005A	TXPDO_BUF_ADRS (RO)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADRS															

The address of the transmit PDO message buffer in the DPRAM memory region.

4.3.1.10 Receive PDO Message Buffer Size Register (RXPDOi_ BUF_SIZE)

0x005C, 0x0060 and 0x0064	RXPDOi_BUF_SIZE (RO)														
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SIZE															

The size of the receive PDO message buffer in Bytes. If RPDOi is not present, the size will be 0.

4.3.1.11 Receive PDO Message Buffer Address Register (RXPDOi_BUF_ADRS)

0x005E, 0x0062 and 0x0066	RXPDOi_BUF_ADRS (RO)														
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADRS															

The address of the receive PDO message buffer in the DPRAM memory region. If RPDOi is not present, the address will be 0.

4.3.1.12 Asynchronous Message Transmit Buffer Size Register (ASYNC_TX_BUF1_SIZE)

0x0068	ASYNC_TX_BUF1_SIZE (RO)														
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SIZE															

The size of the asynchronous transmit[18] message buffer in Bytes.

4.3.1.13 Asynchronous Message Transmit Buffer Address Register (ASYNC_TX_BUF1_ADRS)

0x006A	ASYNC_TX_BUF1_ADRS (RO)														
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADRS															

The address of the asynchronous transmit[18] message buffer.

4.3.1.14 Asynchronous Message Receive Buffer Size Register (ASYNC_RX_BUF1_SIZE)

0x006C	ASYNC_RX_BUF1_SIZE (RO)														
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SIZE															

The size of the asynchronous receive[18] message buffer in Bytes.

4.3.1.15 Asynchronous Message Receive Buffer Address Register (ASYNC_RX_BUF1_ADRS)

0x006E	ASYNC_RX_BUF1_ADRS (RO)														
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADRS															

The address of the asynchronous receive[18] message buffer.

[18] Seen from AP point of view.

4.3.1.16 Asynchronous Message Transmit Buffer Size Register (ASYNC_TX_BUF2_SIZE)

0x0070	ASYNC_TX_BUF2_SIZE (RO)														
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SIZE															

The size of the asynchronous transmit[18] message buffer in Bytes.

4.3.1.17 Asynchronous Message Transmit Buffer Address Register (ASYNC_TX_BUF2_ADRS)

0x0072	ASYNC_TX_BUF2_ADRS (RO)														
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADRS															

The address of the asynchronous transmit[18] message buffer.

4.3.1.18 Asynchronous Message Receive Buffer Size Register (ASYNC_RX_BUF2_SIZE)

0x0074	ASYNC_RX_BUF2_SIZE (RO)														
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SIZE															

The size of the asynchronous receive[18] message buffer in Bytes.

4.3.1.19 Asynchronous Message Receive Buffer Address Register (ASYNC_RX_BUF2_ADRS)

0x0076	ASYNC_RX_BUF2_ADRS (RO)														
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADRS															

The address of the asynchronous receive[18] message buffer.

4.3.1.20 Transmit PDO Acknowledge Buffer (TXPDO_ACK)

0x0080	TXPDO_ACK														
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BUFFER_ACK															

This field is accessible by PCP and AP separately and controls the virtual buffer management.
A write access of either 8 or 16 bit will switch to the next valid (= latest updated) virtual buffer.
If reading the register's content, the value represents the current used virtual buffer. However, it is not necessary to read-back the currently used virtual buffer, only for debugging purpose.
0x0000 … virtual buffer 0
0x1111 … virtual buffer 1
0x2222 … virtual buffer 2

4.3.1.21 Receive PDO Acknowledge Buffer (RXPDO_ACK)

0x0082, 0x0084 and 0x0086	RXPDOi_ACK														
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BUFFER_ACK															

This field is accessible by PCP and AP separately and controls the virtual buffer management.
A write access of either 8 or 16 bit will switch to the next valid (= latest updated) virtual buffer.
If reading the register's content, the value represents the current used virtual buffer. However, it is not necessary to read-back the currently used virtual buffer, only for debugging purpose.

0x0000 … virtual buffer 0
0x1111 … virtual buffer 1
0x2222 … virtual buffer 2
0xFFFF … RXPDOi is not implemented (disabled by generics)

4.3.1.22 Control Register of Synchronous IR Signal (SYNC_IRQ_CTRL)

This register is accessible from PCP and AP side with different content!
AP side:

0x0088			**SYNC_IRQ_CTRL (AP only)**				
7	6	5	4	3	2	1	0
-							IRQ_ACK
0x0089			**SYNC_IRQ_CTRL (AP only)**				
15	14	13	12	11	10	9	8
EN	-						

The IRQ_ACK bit has to be set, if an IRQ was generated by the PCP to the AP. This will acknowledge the IRQ.

Bit	Name	Default	Description
0	IRQ_ACK	0	Self-clearing bit! 1 = acknowledges IRQ
15	EN	0	1 = Interrupt Mode, 0 = Polling Mode (IR signal deactivated)

PCP side:

0x0088			**SYNC_IRQ_CTRL (PCP only)**				
7	6	5	4	3	2	1	0
IRQ_EN	IRQ_MODE	-					IRQ_SET
0x0089			**SYNC_IRQ_CTRL (PCP only)**				
15	14	13	12	11	10	9	8
-							

The IRQ_EN bit enables the IRQ generation for any mode. The IRQ_MODE bit sets the IRQ generation source. The IRQ_SET bit is only available if IRQ_MODE = ‘0’. Otherwise any write to this bit will be ignored by the PDI PCP side!

Bit	Name	Default	Description
7	IRQ_EN	0	0 = IRQ disabled, 1 = IRQ enabled
6	IRQ_MODE	0	0 = IRQ is generated by IRQ_SET, 1 = IRQ is triggered by hardware
0	IRQ_SET	0	Self-clearing bit! 1 = generates IRQ

4.3.1.23 LED Control (LED_CNTRL)

0x0094				**LED_CNTRL**											
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	O1	O0	PA1	PL1	PA0	PL0	E	S

The PCP and AP can read the state of the LEDs in the LED_CNTRL register. If the PCP or the AP writes to this register and the respective force bit is set in LED_CNFG, the LED state is overruled. Note that the AP can overrule the PCP.

Bit	Name	Default	Description
0	S	0	POWERLINK Status LED (1=on, 0=off)
1	E	0	POWERLINK Error LED (1=on, 0=off)

2	PL0	0	Phy 0 link LED (1=on, 0=off)
3	PA0	0	Phy 0 activity LED (1=on, 0=off)
4	PL1	0	Phy 1 link LED (1=on, 0=off)
5	PA1	0	Phy 1 activity LED (1=on, 0=off)
6	O0	0	Optional LED0 (1=on, 0=off)
7	O1	0	Optional LED1 (1=on, 0=off)

4.3.1.24 LED Configuration (LED_CNFG)

0x0096								LED_CNFG							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-								LED_FORCE_EN0							

0x0096				LED_FORCE_EN0			
7	6	5	4	3	2	1	0
FO1	FO0	FPA1	FPL1	FPA0	FPL0	FE	FS

The LED bits set in LED_CNTRL are only written to the outputs if the FORCE_LED register is set appropriately with the corresponding bit pattern. A force enable bit set to “1” of the AP will overwrite any related bit set by the PCP in the LED_CNTRL.

Bit	Name	Default	Description
0	FS	0[19]	Force enable of POWERLINK Status LED (1=on, 0=off)
1	FE	0[19]	Force enable of POWERLINK Error LED (1=on, 0=off)
2	FPL0	0	Force enable of Phy 0 link LED (1=on, 0=off)
3	FPA0	0	Force enable of Phy 0 activity LED (1=on, 0=off)
4	FPL1	0	Force enable of Phy 1 link LED (1=on, 0=off)
5	FPA1	0	Force enable of Phy 1 activity LED (1=on, 0=off)
6	FO0	0	Force enable of Optional LED0 (1=on, 0=off)
7	FO1	0	Force enable of Optional LED1 (1=on, 0=off)

4.4 I/O Port (SMP)

Every write and read access to/from the first 4 bytes will access the PORT registers only. The I/O Port direction is configured via the x_pconfig input port of the IP-core.
If one writes data to a port configured as input, the written data is ignored!
If one reads data from a port configured as output, the read data must be ignored!

SMP write access

SMP (WR)																															
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x0000	PORTOUT3								PORTOUT2								PORTOUT1								PORTOUT0							
0x0004	PORTEXTRA																															

[19] This bit is only for PCP side set to “1” per default.

Writing to one of the four PORTOUT registers will simply output the data pattern to the port if configured as output.

0x0007	**PORTEXTRA**						
7	6	5	4	3	2	1	0
PLK_OP	-	-	-	-	-	-	-

Bit	Name	Default	Description
7	PLK_OP	0	Links to the operational flag of the direct I/O interface. Set/reset this bit to set/reset the operational pin. Read from this bit to obtain the pin's state.

SMP read access

SMP (RD)																													
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x0000	PORTIN3								PORTIN2								PORTIN1								PORTIN0							
0x0004	PORTEXTRA								-								-								PORTDIR							

Reading from one of the four PORTIN registers will read the last latched data pattern if configured as input.

0x0004	**PORTDIR**						
7	6	5	4	3	2	1	0
-				PORTDIR3	PORTDIR2	PORTDIR1	PORTDIR0

Bit	Name	Default	Description
0	PORTDIR0	0	0 = Port is Output, 1 =Port is Input
1	PORTDIR1	0	0 = Port is Output, 1 =Port is Input
2	PORTDIR2	0	0 = Port is Output, 1 =Port is Input
3	PORTDIR3	0	0 = Port is Output, 1 =Port is Input

5 Definitions and Abbreviations

AP	Application Processor
BUF	MAC-internal packet buffer
CMD	Command
CMP	Compare Unit
CN	POWERLINK Controlled Node
DPR	Dual Ported RAM
DPRAM	Dual Ported RAM
FF	Flip Flop
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
GUI	Graphical User Interface
IP	Intellectual Property
IRQ	Interrupt Request
MII	Media Independent Interface
MN	POWERLINK Managing Node
PCB	Printed Circuit Board
PCP	Powerlink Communication Processor
PDI	Process Data Interface
PDO	Process Data Object
PReq	Poll Request (POWERLINK frame type)
PRes	Poll Response (POWERLINK frame type)
RD	Read
RDY	Ready
RMII	Reduced Media Independent Interface
RO	Read Only
SDO	Service Data Object
SMI	Serial Management Interface (Ethernet phy register access)
SMP	Simple I/O Port
SoA	Start of Asynchronous (POWERLINK frame type)
SoC	Start of Cyclic (POWERLINK frame type)
SPI	Serial Peripheral Interface
SYNC	Synchronization (Clock Domain Crossing)
WR	Write

6 References

[1] Wikipedia: “Serial Peripheral Interface Bus”
http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus
access: 2010-09-06

[2] Altera: “Avalon Interface Specification”
http://www.altera.com/literature/manual/mnl_avalon_spec.pdf
Version 11.0 (May 2011)

[3] FDS – openPOWERLINK Slave DevKit – Time Synchronization
Bernecker + Rainer
October 25, 2011

7 Figure Index

8 Table Index

9 Index