POWERLINK

IP-Core

Date: September 7, 2011

Project Number: AT-B0-000002

I Versions

Version	Date	Comment	Edited by
0.1	Aug 30, 2010	First Edition	Zelenka Joerg
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 Avalon 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 Avalon Memory Mapped Master	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

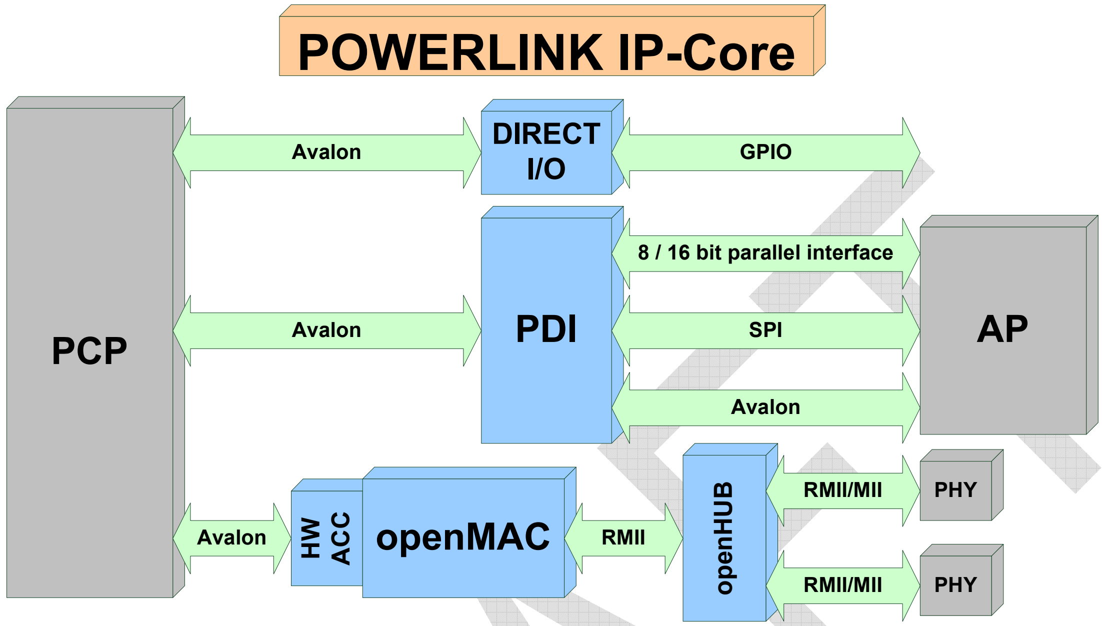

Fig. 1: POWERLINK IP-Core Block Diagram

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 (e.g. Altera SOPC)
- 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)

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.

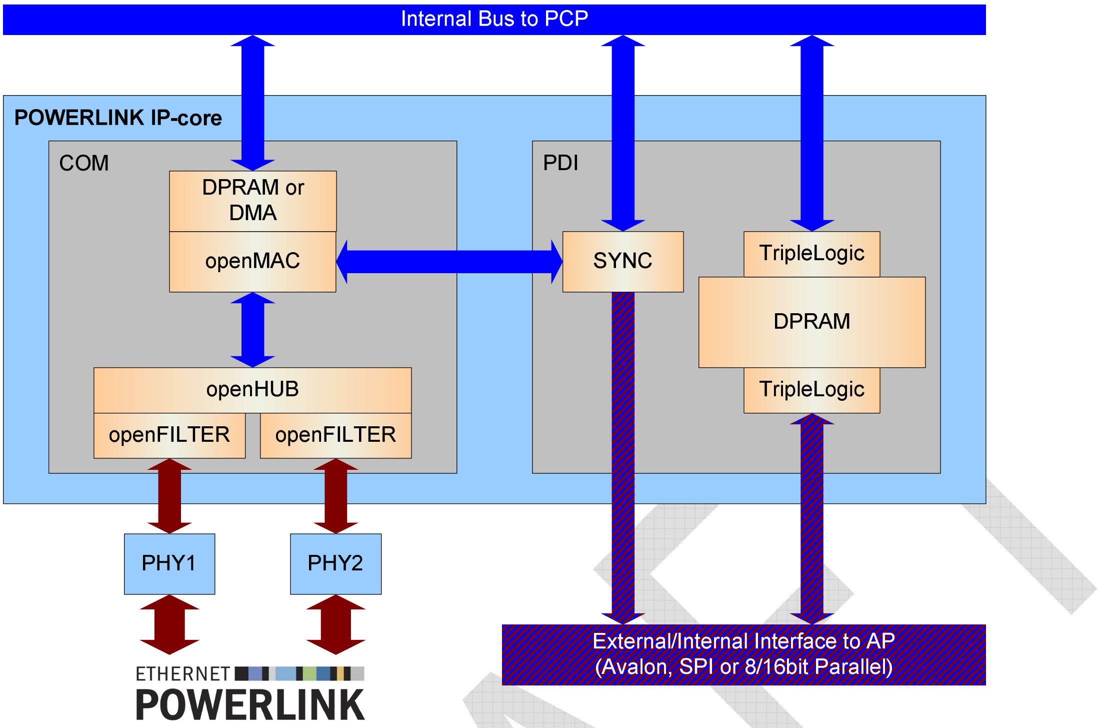

Fig. 2: POWERLINK IP-core System Overview Configuration Case 1

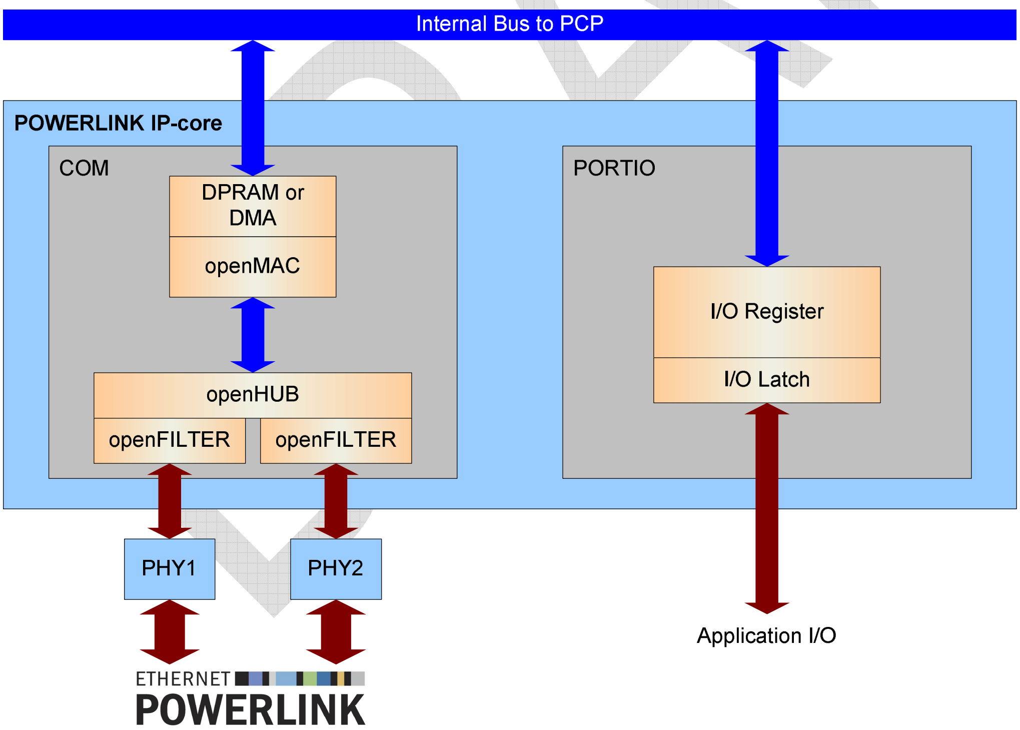

Fig. 3: POWERLINK IP-core Sysrem Overview Configuration Case 2

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
  - Nios II
  - Avalon bus
- 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 (e.g. Nios II)	✓	✓[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 the Avalon bus fabrication.
[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 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 some 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	Packet buffer memory clock	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_Altera.vhd)
- openMAC DMA master for Avalon (openMAC_DMAmaster.vhd)
  - Avalon Memory Mapped Master handling logic (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 Avalon MM Master
- RX and TX via Avalon MM Master

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.

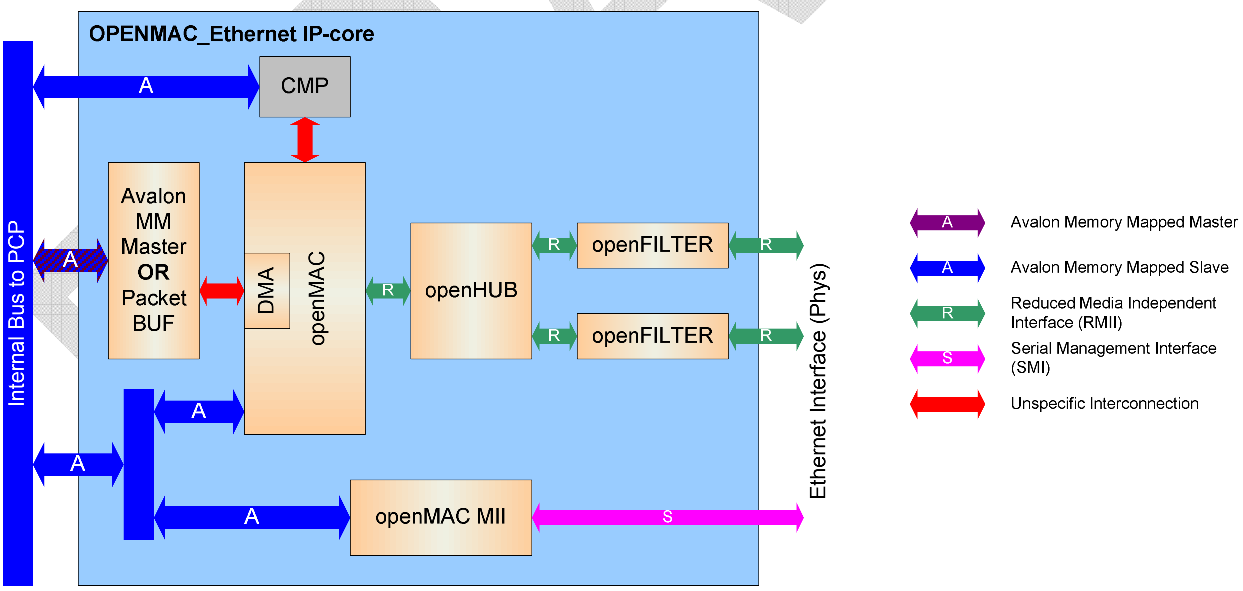

Fig. 4: OpenMAC IP-core – overview

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” (docu_openMAC.pdf)!

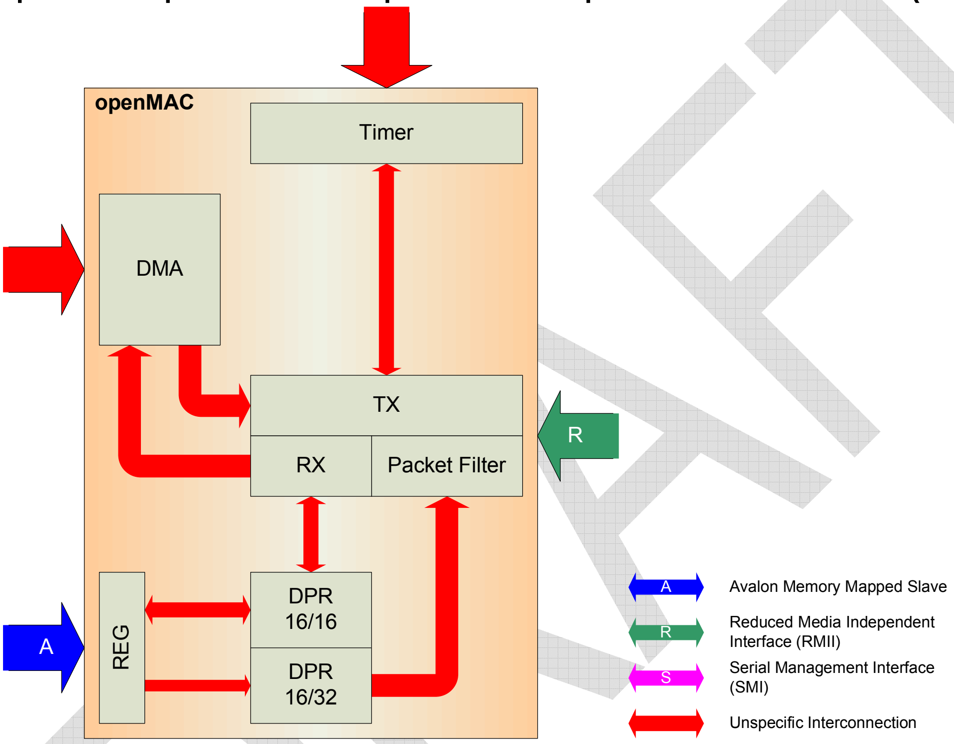

Fig. 5: OpenMAC Block diagram

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

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.

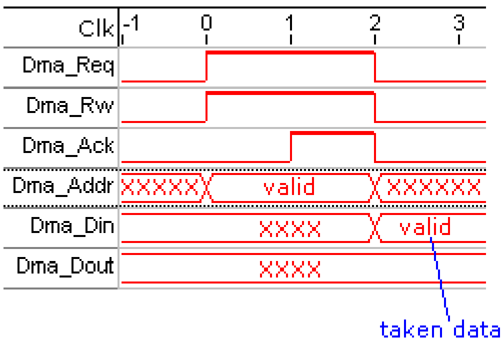

Fig. 6: DMA read transfer

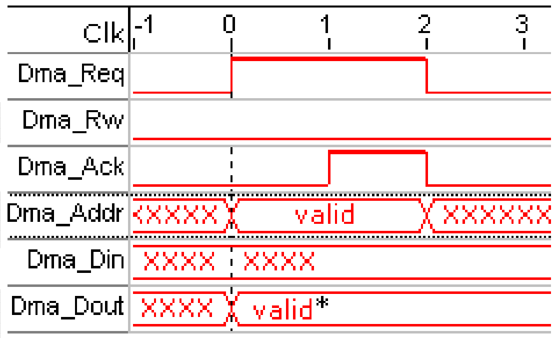

Fig. 7: DMA write transfer[8]

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!

3.2.2 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 auto-response 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.

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

The usage of the packet buffer is optional, depending on the generic set (useIntPacketBuf_g and userx-IntPacketBuf_g).

3.2.3 Avalon 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 (embedded memory blocks, M9K), which enables smaller chip sizes. However, it has to be ensured that the memory's read latency is static, which is valid for embedded memory blocks (M9K) 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 Avalon MM Master logic is able to transfer data with single beat or burst transfers, which has to be determined in the SOPC (generics).

Important:

Only use the Avalon MM Master interface if the connected memory is either M9K or external SRAM (10 ns). Also consider to use only 3 Avalon MM Masters[9] beside the MAC_DMA.
If the connected memory device is e.g. SDRAM, it is not recommended to store TX packets in that kind of memory.[10] 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 Avalon MM Master logic is implemented in the file openMAC_DMAmaster.vhd and is instantiated by the openMAC_Ethernet component depending on the generics defined by the SOPC user. 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 automatically inserted clock domain crossing logic (CDC) or clock crossing bridges inserted by the user in SOPC.
The architectural design is visualized in Fig. 8, which assumes RX and TX packet transfers via the Avalon MM 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

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 (FIFO underflow). In the opposite direction (RX), the master handler tries to empty the RX FIFO continuously, which avoids FIFO overflow.

[9] For instance: Nios II data and instruction master, DMA and the openMAC DMA
[10] Dynamic RAM types have to refresh their data content, which introduces long access latency to the host (e.g. CPU or DMA).

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 size (note that the configuration in SOPC is limited to values that are optimized to save resources).
The master handler uses one link for write- and read-transfers, to reduce the complexity of the Avalon 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 Avalon MM 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.

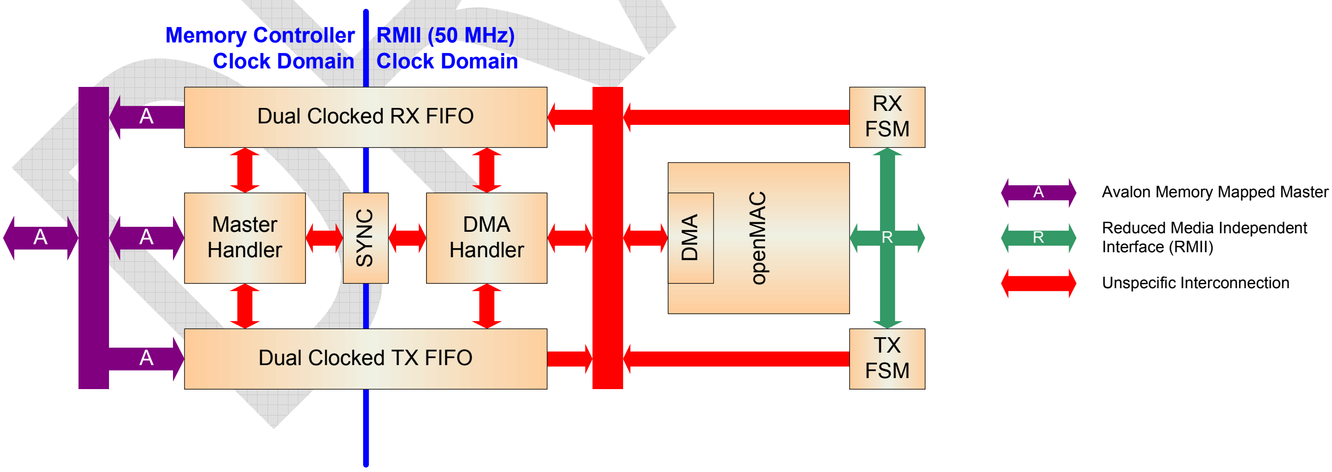

Fig. 8: Block diagram Avalon Memory Mapped Master

3.2.3.2 Performance Consideration

TBD

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[11], 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")

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

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).

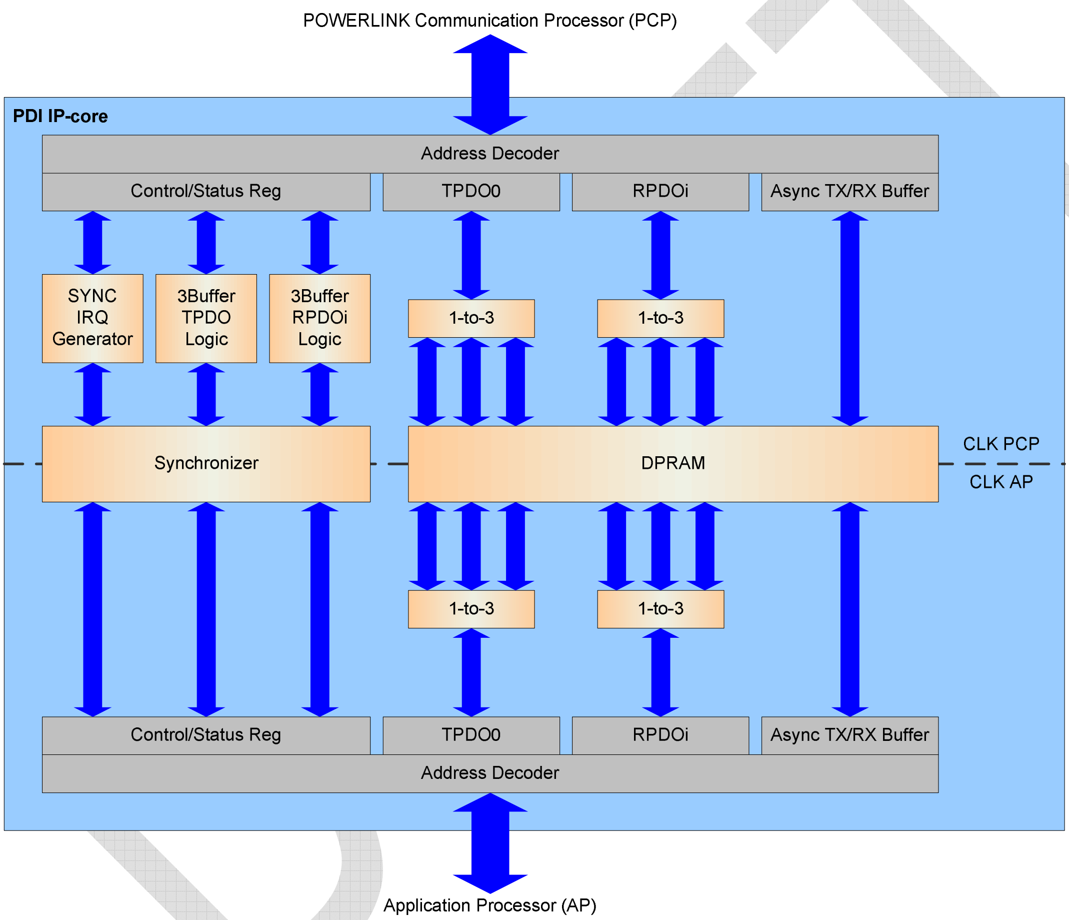

Fig. 9: PDI IP-core architecture overview

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[12]. 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.
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.

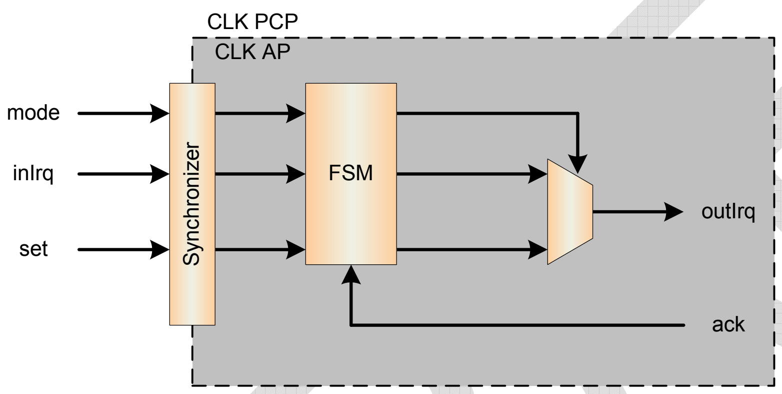

Fig. 10: SYNC IRQ Generator

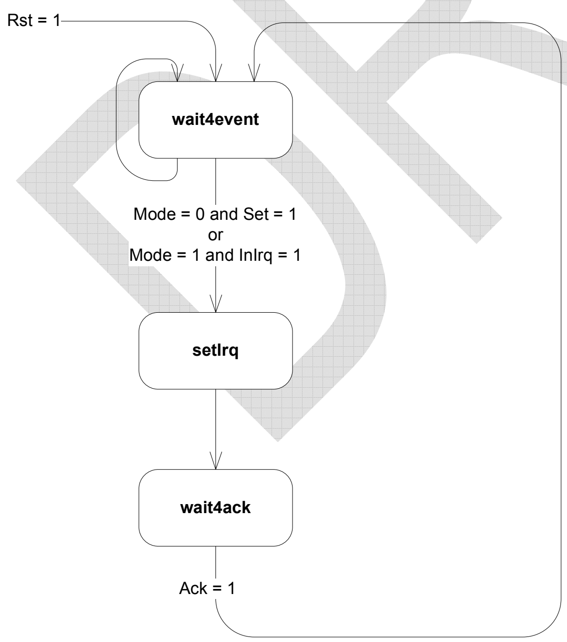

Fig. 11: SYNC IRQ Generator - FSM

---

12 Refer to 4.2.1

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.

Obviously, a two-stage synchronizer used for synchronizing several bits or even bytes require many LEs in an FPGA, thus, the usage of different clock domains should be avoided. Out of this SOPC restricts the PDI to one clock domain (50MHz) valid for the PCP and AP. SOPC automatically inserts clock domain crossing logic (CDC) at the Avalon Slave entry points, however, the resource usage is of course lower using this approach.

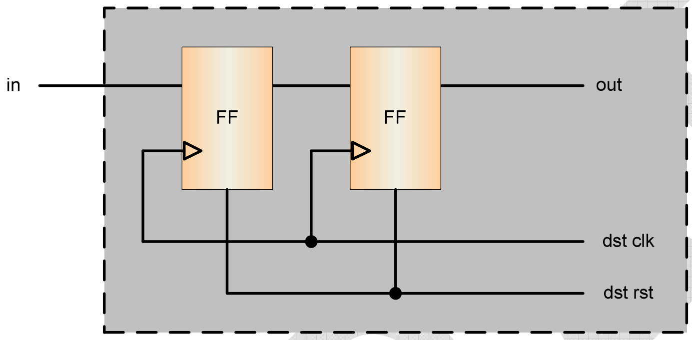

Fig. 12: Simple 2-stage Synchronizer

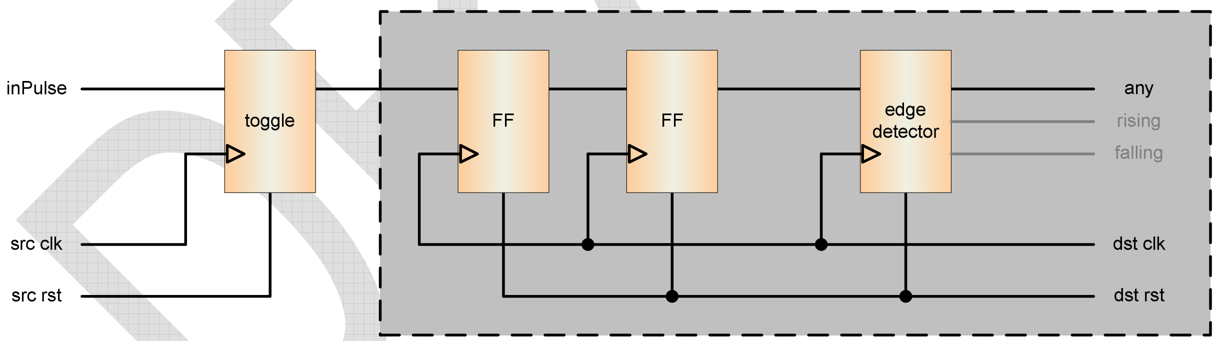

Fig. 13: 2-stage Synchronizer for pulse transfer

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.

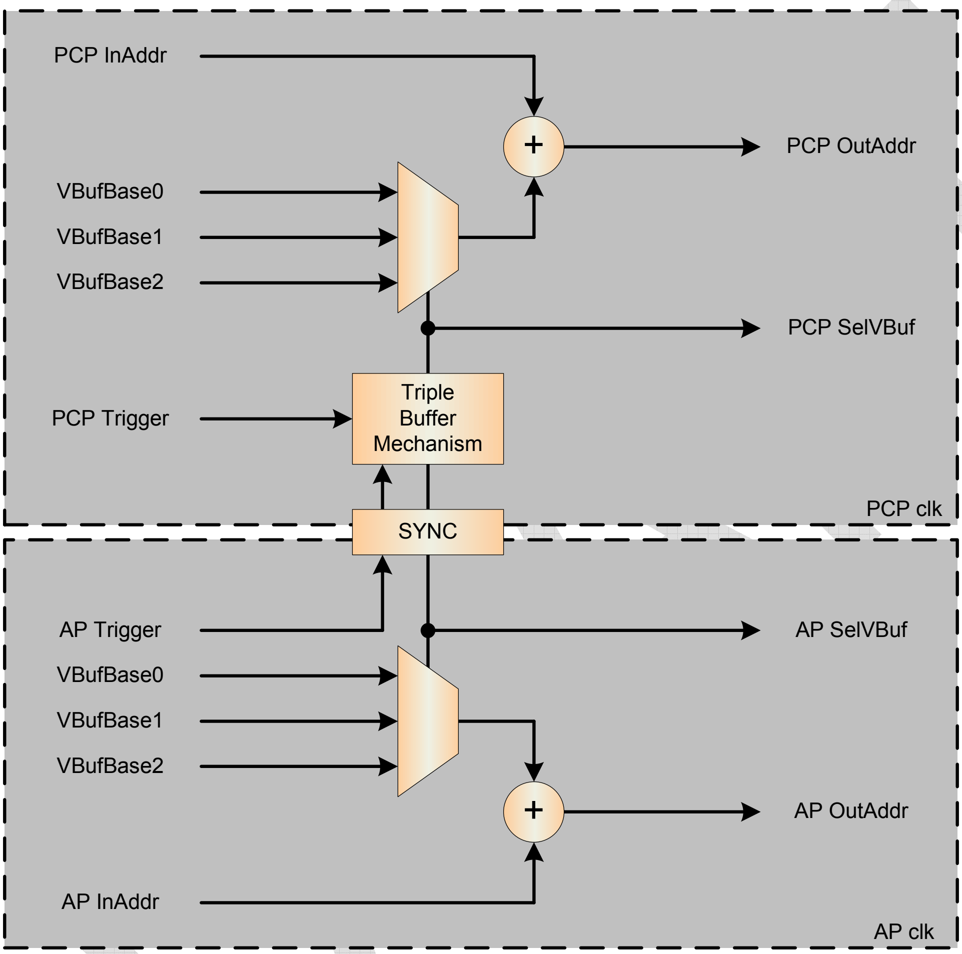

Fig. 14: Triple Buffer Logic System Overview

Variable[13] 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.

[13] The expression "VARIABLE" used in this context is not equal to a variable used in VHDL.

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
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[14] 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.

3.4 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.

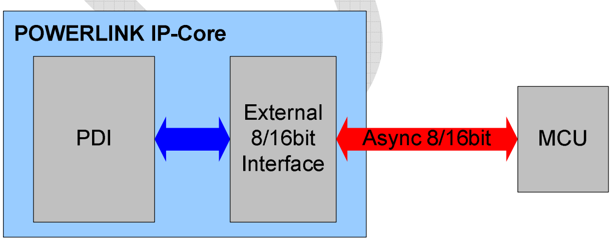

Fig. 15: Asynchronous 8/16bit Parallel Interface Overview

The external interface is mapped to the Avalon bus signals of the PDI, but this interconnection does not use the Avalon Bus fabrication. Thus from the MCU's point of view the PDI works like a SRAM component with a predefined timing specification.

[14] The expression "variable" used in this context is equal to the variable usage in VHDL.

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 Avalon bus signals. However, the Avalon bus specification is not considered, since the Asynchronous 8/16bit Parallel Interface IP-core should be directly connected to the PDI AP side. If this IP is used with the Avalon Bus fabrication, errors may be present since “waitrequest” is not implemented!
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.

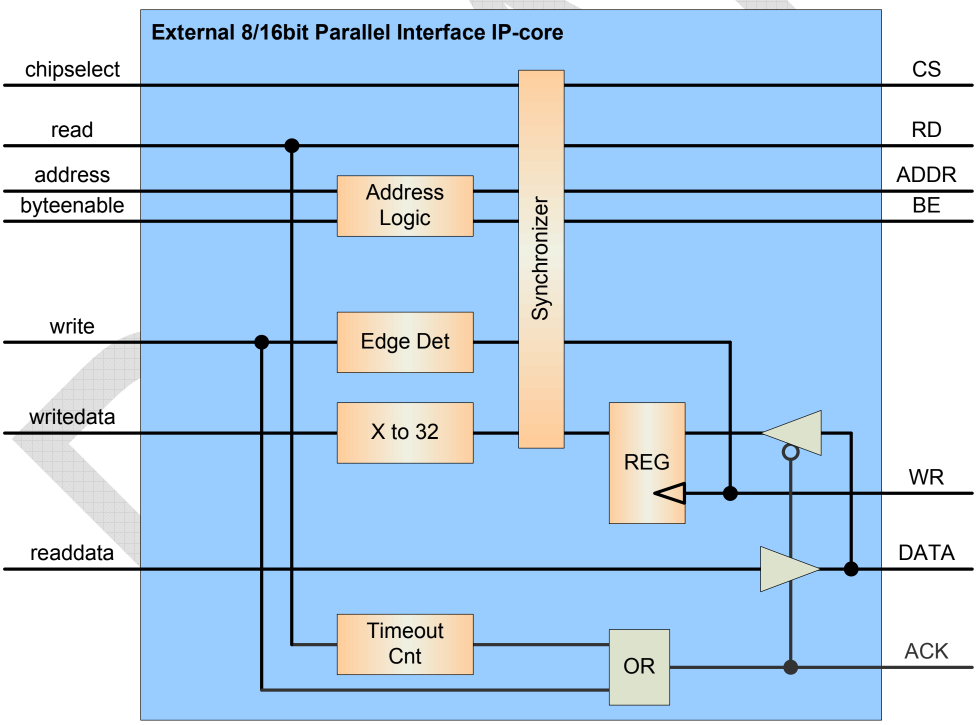

Fig. 16: Asynchronous 8/16bit Parallel Interface IP-core

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_{HA}	40ns	-	The valid address has to be hold before the next write or read access.
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_{RNA}	40ns	-	After a read access the next write or read has to be asserted after a delay.
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_{AP}	20ns	-	ACK signal assertion duration

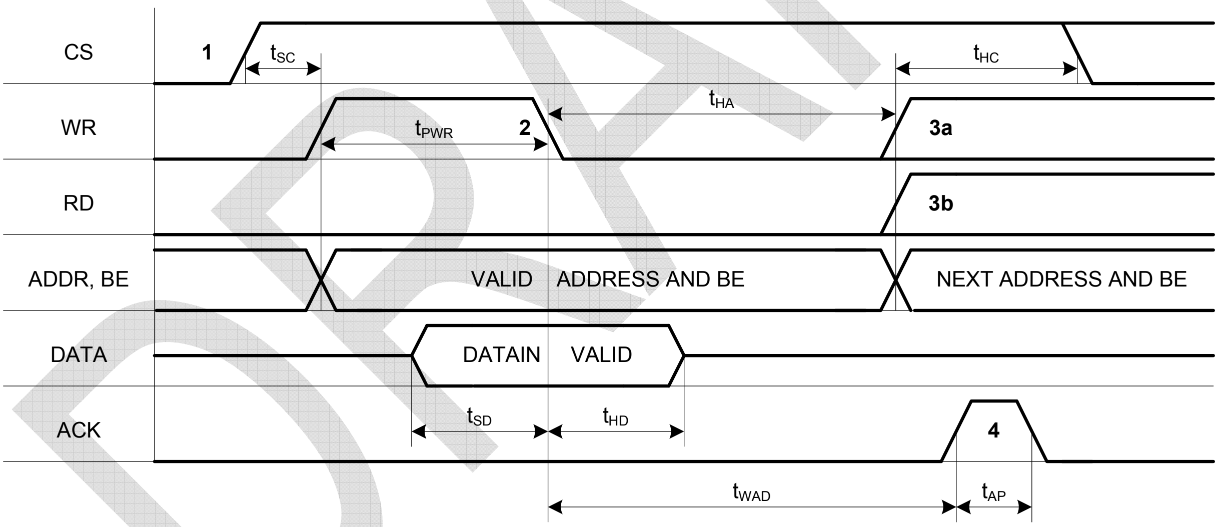

Fig. 17: Write Access Timing

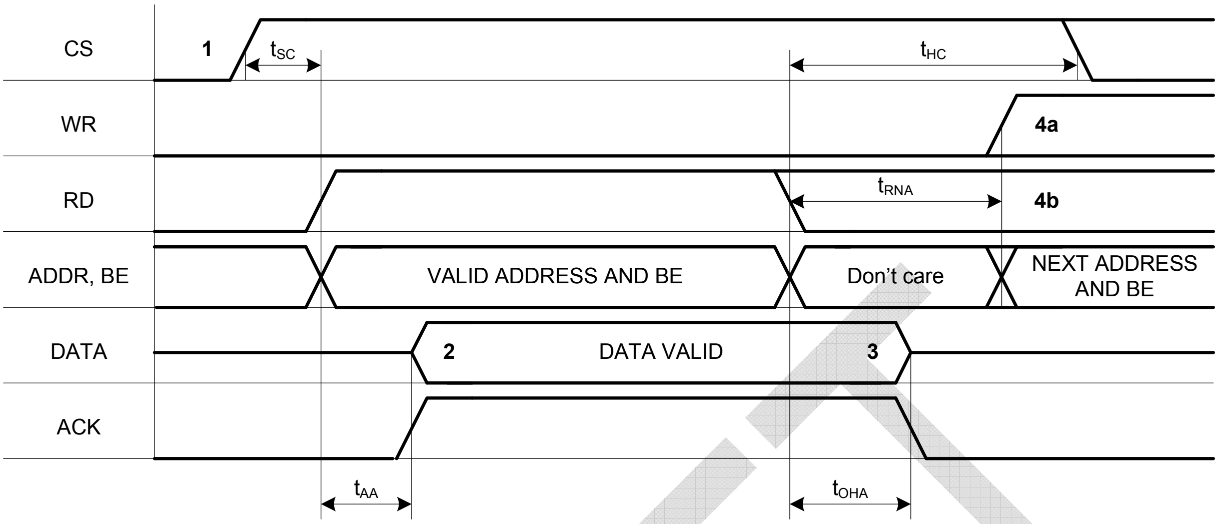

Fig. 18: Read Access Timing

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

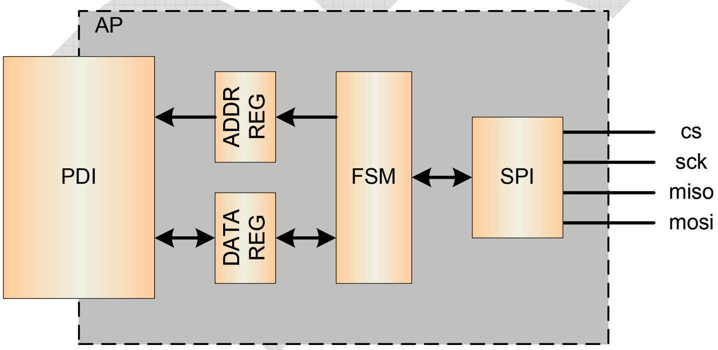

Fig. 19: SPI IP-core architecture overview

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.

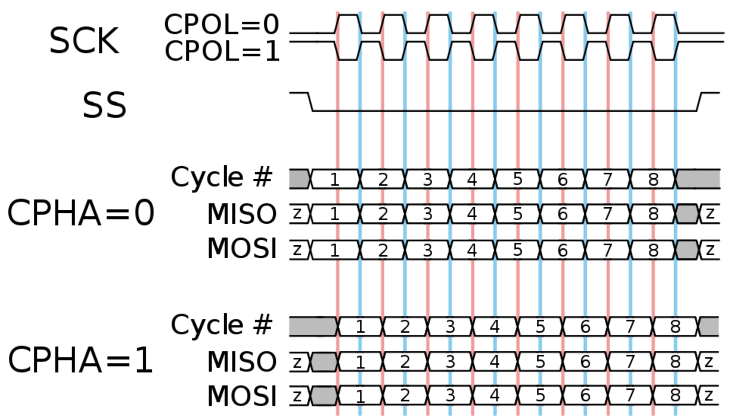

Fig. 20: SPI timing [1]

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[15] 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**

[15] 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!

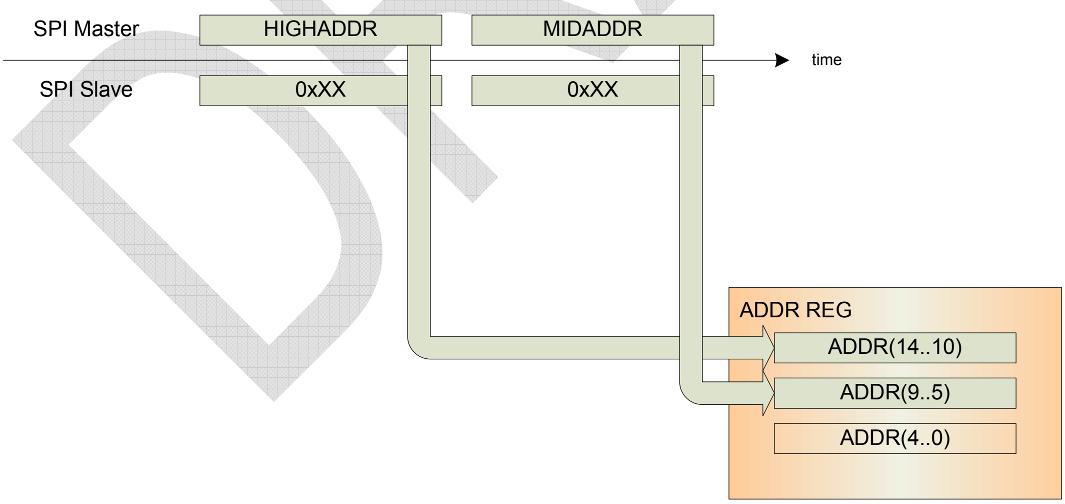

Fig. 21: SPI Data Flow – Addressing

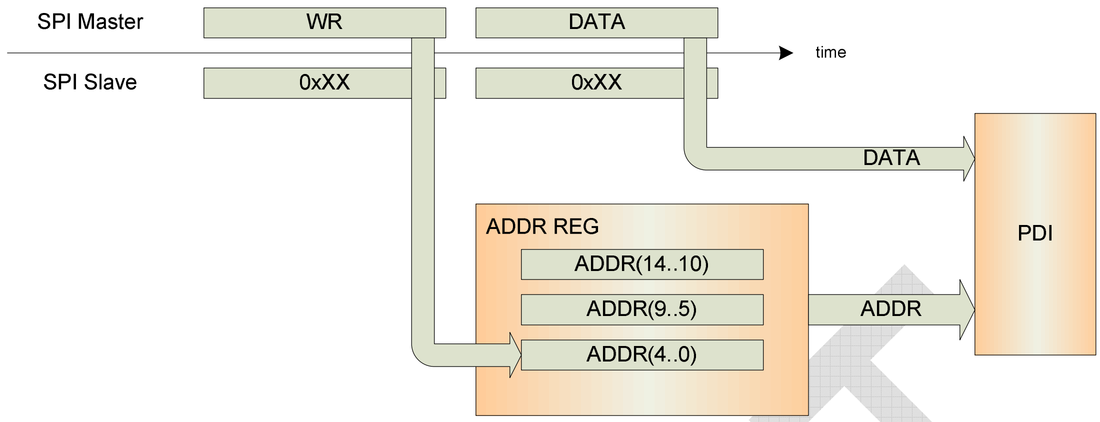

Fig. 22: SPI Data Flow – Write Data

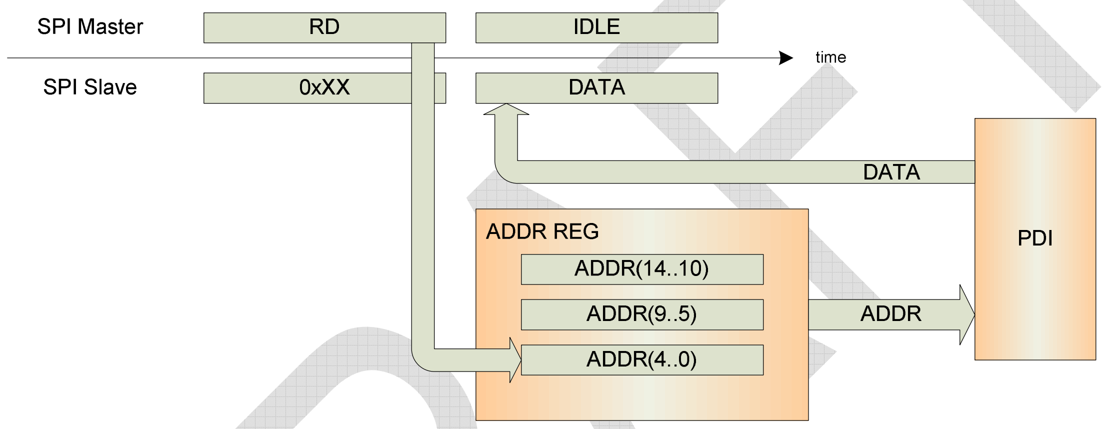

Fig. 23: SPI Data Flow – Read Data

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.

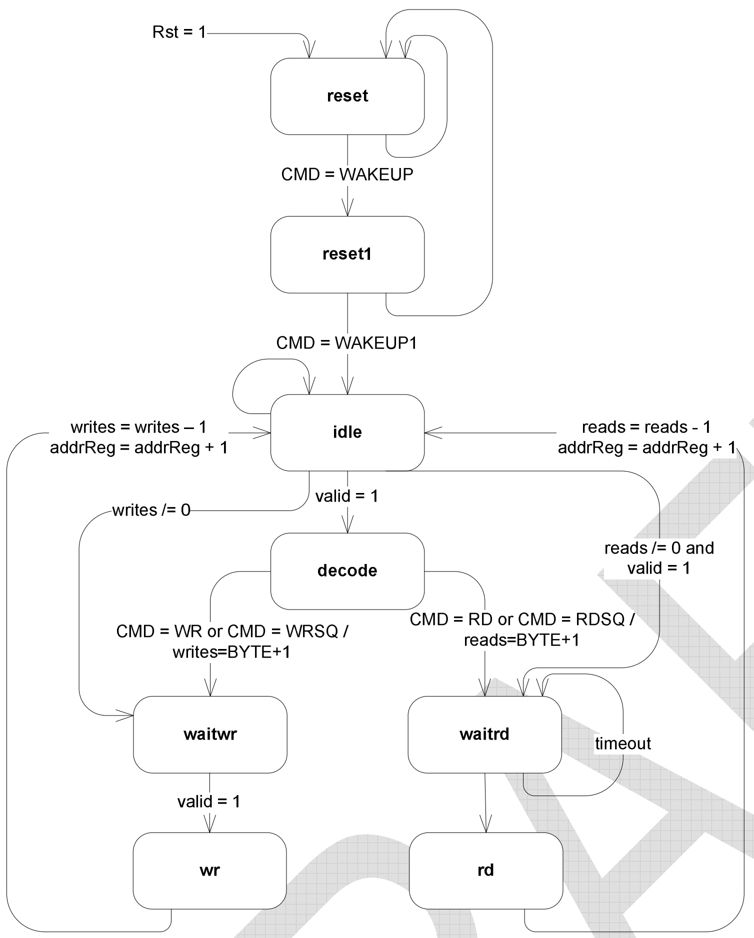

Fig. 24: SPI FSM

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 Avalon Memory Mapped Slave connections. Refer to Tab. 7 and its footnotes.

Tab. 7: Available Memory Mapped Interfaces

Nr	Configuration	openMAC[16]			PDI		PORTIO
	Memory Mapped Connection Points[17]	CMP	REG	BUF	PCP	AP	SMP
1	Simple I/O (no AP)	✓	✓	✓	×	×	✓
2	Internal AP (e.g. Nios II)	✓	✓	✓	✓	✓	×
3	External AP (e.g. MCU or DSP) for high data throughput	✓	✓	✓	✓	✓[18]	×
4	External AP (e.g. MCU or DSP) for low data throughput	✓	✓	✓	✓	✓[19]	×

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 Avalon 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 Avalon Memory Mapped Slave Interfaces connected to the internal Avalon bus. The AP port to the PDI uses a special 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 (4.2.3).

---

16 OpenMAC is mandatory for every configuration.
17 The expression “Memory Mapped Connection Point” denotes independent regions on the internal Avalon bus.
18 For high data throughput to FPGA-external components it is recommended to use the parallel 8/16bit interface configuration.
19 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 byte pattern 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 has to be performed.

CMP write access

<table>
<tr><th colspan="33">CMP (WR)</th></tr>
<tr><td></td><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>0x0008</td><td colspan="32">CMP_VAL_TOG</td></tr>
<tr><td>0x0004</td><td colspan="8">-</td><td colspan="8">-</td><td colspan="8">-</td><td colspan="8">CMP_CNTR</td></tr>
<tr><td>0x0000</td><td colspan="32">CMP_VAL_IRQ</td></tr>
</table>

<table>
<tr><th colspan="2">0x0004</th><th colspan="6">CMP_CNTR</th></tr>
<tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td colspan="3">-</td><td>EN_TOG</td><td colspan="3">-</td><td>EN_IRQ</td></tr>
</table>

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

CMP read access

<table>
<tr><th colspan="33">CMP (RD)</th></tr>
<tr><td></td><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>0x0008</td><td colspan="32">CMP_VAL_TOG</td></tr>
<tr><td>0x0004</td><td colspan="8">-</td><td colspan="8">-</td><td colspan="8">-</td><td colspan="8">CMP_STAT</td></tr>
<tr><td>0x0000</td><td colspan="32">MAC_TIME</td></tr>
</table>

<table>
<tr><th colspan="2">0x0000</th><th colspan="6">CMP_STAT</th></tr>
<tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td colspan="2">-</td><td>TOG</td><td>EN_TOG</td><td colspan="2">-</td><td>IRQ</td><td>EN_IRQ</td></tr>
</table>

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
4	EN_TOG	0	0 = CMP TOG is disabled, 1 = CMP TOG is enabled
5	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

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)

TBD

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

<table>
<tr><th colspan="33">SMP (WR)</th></tr>
<tr><td></td><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>0x0000</td><td colspan="8">PORTOUT3</td><td colspan="8">PORTOUT2</td><td colspan="8">PORTOUT1</td><td colspan="8">PORTOUT0</td></tr>
<tr><td>0x0004</td><td colspan="8">PORTEXTRA</td><td colspan="8"></td><td colspan="8"></td><td colspan="8"></td></tr>
</table>

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

<table>
<tr><th colspan="2">0x0007</th><th colspan="6">PORTEXTRA</th></tr>
<tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>PLK_OP</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td></tr>
</table>

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

<table>
<tr><th colspan="33">SMP (RD)</th></tr>
<tr><td></td><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>0x0000</td><td colspan="8">PORTIN3</td><td colspan="8">PORTIN2</td><td colspan="8">PORTIN1</td><td colspan="8">PORTIN0</td></tr>
<tr><td>0x0004</td><td colspan="8">PORTEXTRA</td><td colspan="8">-</td><td colspan="8">-</td><td colspan="8">PORTDIR</td></tr>
</table>

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

4.5 System description (system.h)

The POWERLINK IP-core is included into an SOPC (Altera) or any other processor environment. In order to provide vital information to the PCP a header file is used (e.g. system.h). The POWERLINK IP-core provides the following defines:

Tab. 9: System description parameters

Name	Macro	Value
MAC-internal buffer size	MACBUFSIZE	Any integer
MAC-internal RX buffer size	MACRXBUFSIZE	Any integer
MAC-internal RX buffers	MACRXBUFFERS	Any integer
MAC-internal TX buffer size	MACTXBUFSIZE	Any integer
MAC-internal TX buffers	MACTXBUFFERS	Any integer
Number of PDI RPDOs	PDIRPDOS	1, 2 or 3
Number of PDI TPDOs	PDITPDOS	1
POWERLINK IP-core configuration	CONFIG	0, 1, … , 5
AP interface endianness	CONFIGAPENDIAN	0 or 1

Two parameters mentioned in Tab. 9 must be discussed furthermore.

Tab. 10: System description parameters, details

Macro	Value	Description
CONFIG	0	Direct I/O (no AP intended)
	1	8bit parallel interface
	2	16bit parallel interface
	3	SPI
	4	Avalon bus
	5	openMAC only (no PDI available)
CONFIGAPENDIAN	0	Little endian
	1	Big endian

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)

7 Figure Index

8 Table Index

9 Index