SysKonnect

SysKonnect
a business unit of
Schneider & Koch & Co.
Datensysteme GmbH

Siemensstraße 23
D-76275 Ettlingen
Tel. 0 72 43 / 50 2100
Fax 0 72 43 / 50 29 89

November 6, 1998
Version 1.0

SK-NET GENESIS <x>
Technical Manual

SK internal documentation

Revision History

Revision	Released at	Changes from Previous Release	Remarks
1.0	November 6, 1998		

Related Documentation

“PCI Local Bus Specification”, Revision 2.1, June1, 1995

“PCI Local Bus Specification”, Review Draft Revision 2.2, June 8, 1998

“PCI Bus Power Management Interface Specification”, Review Draft Revision 1.1 May 4, 1998

“PCI Bus Hot Plug Specification” Review Draft Revision 1.0, June 15, 1997

“ISO/IEC 8802-3 : 1993 [ANSI/IEEE Std 802.3, 1993 Edition] Information technology - - Local and metropolitan area networks - Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications”

“IEEE 802.3z-1998 IEEE Supplement to ISO/IEC 8802-3 : 1996 Physical Layer, Repeater and Management Parameters for 1 000 Mb/s operation” Approved Draft (former IEEE 802.3z/D5)

“IEEE 802.3x&y-1997 Supplement to ISO/IEC 8802-3 : 1996 Specification for 802.3 Full Duplex Operation and Physical Layer Specification for 100 Mb/s Operation on Two Pairs of Category 3 or Better Balanced Twisted Pair Cable (100BASE-T2): Parts 1 and 2”

“IEEE Draft P802.3ab/D4.0 Supplement to Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method & Physical Layer Specifications : Physical Layer specification for 1 000 Mbps operation on four pairs of Category 5 or better balanced twisted pair calbe (1 000 BASE-T)”

“XaQti XQ11800 FP 1 000 Mbps Gigabit Ethernet Controller” Data Sheet Order Number: 11800-0198-07 Applies to XQ11800FP chip revisions B.2 and earlier Rev. 7 February, 1998
and
“Differences between Rev. B2 & Rev. C XMAC II”

“Am79761 Physical Layer 10-Bit Transceiver for Gigabit Ethernet (GigaPHY^{TM}-SD)” Preliminary Data Sheet from Advanced Micro Devices (AMD) Publication #21560

“HDMP 1636, HDMP 1646 Gigabit Ethernet Transceiver” Technical Data from Hewlett-Packard Co. 5966-0683E (9/97)

“MDX-19-4-2-X-X Optical Gigabit Ethernet Transceiver” OE8608-8140 from Method Electronics, Inc.

“AC/AC PECL (5V/3.3V) V23826-K15-C73/C373 Single Mode 1300 nm 1.3 Gigabit Ethernet 1x9 Transceiver” July 1998 Preliminary from Siemens Semiconductor Group

“LM80 Serial Interface ACPI-Compatible Microprocessor System Hardware Monitor” DS100040 from National Semiconductor Corporation

“LT1086 Series 1,5 A Low Dropout Positive Regulators, Adjustable and Fixed 2.85V, 3.3V, 3.6 V, 5V, 12V” from Linear Technology

Current Release

Approved by		
Hardware Engineer		
Software Engineer (if applicable)		
Technical Writer		

Document Conventions

Throughout this document tables will be used to present registers and memory maps They are used to list the usage and settings of registers, of single bits in registers, etc. To keep tables more compact some abbreviations, special notation will be used:

Register Descriptions		
Write:		
	ne	no effect
	yes	writable
	sh	special handling as described in the text or the table
	exec	execution of this command - may refer to single bits - the command is executed if the corresponding bit is written as 1.
Read:		
	aw	as written
	value	as defined by itself
Commands Commands are executed, if the appropriate bit is set. Read value as defined.		
Exclusive Commands (e.g. Start/Stop, ON/OFF (usually two bits) Commands are executed, if appropriate bit is set to 1. Setting both commands to 1, has no effect. Status is readable: 0b01 or 0b10.		

Especially in register-, signal-, or flag names the following abbreviations may be used:

SM:	state machine
RX:	receive
TX:	transmit
word:	16-bit word
dword:	32-bit word

qword:	64-bit word
EOF:	EndOfFrame
STF:	STartOfFrame
frame:	packet
packet:	frame (are used synonymously)

Table of Contents

Appendix A

Appendix B

SysKonnect

November 6, 1998
Version 1.0

SK-NET GENESIS <x>

Technical Manual

This manual describes the hardware of the network interface card "**SK-NET**[1] **GENESIS <x>**". It contains a complete description of the card's programming interface. This will enable the software engineer to develop driver and diagnosis software according to the specifications mentioned below.

1.0 Introduction

The "**SK-NET GENESIS <x>**" is a highly integrated Gigabit Ethernet adapter for PCI LOCAL BUS systems designed to address high-performance systems' requirements. The bus-master architecture with integrated buffer management provides high throughput besides low CPU and system bus load.

---

1. SK, SK-NET are trademarks of Schneider & Koch & Co. Datensysteme GmbH and SysKonnect, Inc. All other brand or product names are trademarks or registered trademarks of their respective holders.

SK-NET GENESIS <x> is an interim name. In the market the NICs will be known as SK-<xxxx>, where <xxxx> is a four digit number.

1.1 Product Highlights

The "**SK-NET GENESIS <x>**" is a Gigabit Ethernet adapter based on an ASIC from SysKonnect and an Gigabit Ethernet Controller (XMACII) from XaQti Corporation.

The ASIC provides an interface between the PCI based host system and the Gigabit Ethernet controller with on-board buffer memory. The XMACII Gigabit Ethernet controller provides control and access to the network. "**SK-NET GENESIS <x>**" products will be available in different versions: with one XMAC[1] (single link) or two XMACs (dual link). The dual link version provides data path redundancy and improved network availability.

1.1.1 ASIC

- ❏ Compliant to PCI specification/Rev. 2.1 (2.0) and ECRs for Rev. 2.2 (Power Management, VPD, New Capabilities)
- ❏ Universal I/O - 3.3V and 5V signals
- ❏ 64-bit bus master interface for data transfer, 32-bit bus target interface for control/status
- ❏ Supporting full transfer rate of 528MBytes/s between PCI local bus and FIFOs.
  Supporting advanced PCI protocol (Memory Write And Invalidate, Memory Read Line and Multiple for cache line sizes up to 128 32-bit-words)
- ❏ High performance bus master architecture with integrated Buffer Management Unit (BMU) for low CPU and bus utilization.
- ❏ Individual 64-bit wide FIFOs for each receive and transmit queue, providing buffering between PCI local bus and buffer memory. The buffer memory supports bursts of maximum 199 × 64-bit-words length at 66 MHz bus clock.
- ❏ Supporting misaligned data transfers between system memory and FIFOs.
- ❏ Support for byte reordering for descriptors in big endian systems. (Data is assumed to be little endian on the PCI bus)
- ❏ Interrupt moderation (PacedPacketBatch)
- ❏ Rate control on second ('synchronous') transmit queue
- ❏ TCP/IP checksum generation/checking
- ❏ Parity generating and checking at PCI local bus interface, internal data paths and external RAM data paths
- ❏ Advanced power management, supporting power modes D0 and D3
- ❏ Three independent queues for each link: receive, transmit/asynchronous and transmit/synchronous

1. MAC means Medium Access Controller in general. A MAC implements the Medium Access Control specification of a network protocol such as Ethernet or Gigabit Ethernet. XMACII is a product from XaQti Corporation implementing the Gigabit Ethernet MAC specification. Usually one speaks about the "single MAC" NIC and the "dual MAC" NIC.

1.1.2 XMAC

- ❏ Supports Gigabit Ethernet IEEE 802.3z requirements and complies with IEEE 802.3x for frame-based flow control and full-duplex operation
- ❏ On-chip transmit/receive FIFOs and 8B/10B PCS Encoder/Decoder
- ❏ On-chip CAM for address matching
- ❏ Full duplex / half duplex modes of operation
- ❏ Jumbo frame handling
- ❏ Link autonegotiation
- ❏ Virtual LAN support
- ❏ SNMP and RMON monitoring

1.1.3 Transceiver and SERDES

The Gigabit Ethernet transceivers are connected with an SERializer/DE-Serializer (SERDES) that translates the 10-bit wide data at the MAC interface to a serial data stream which is sent over the Gigabit Ethernet cabling either fiber optic or copper media via the transceiver.

The "**SK-NET GENESIS <x>**" is designed to attach to several transmission media depending on the type of the mounted transceiver. The design intention is to make the NIC attachable to physical media specified for 1000Mbps operation in the IEEE 802.3 framework. Currently the NIC supports

- ❏ 1000 Base-SX
- ❏ 1000 Base-LX

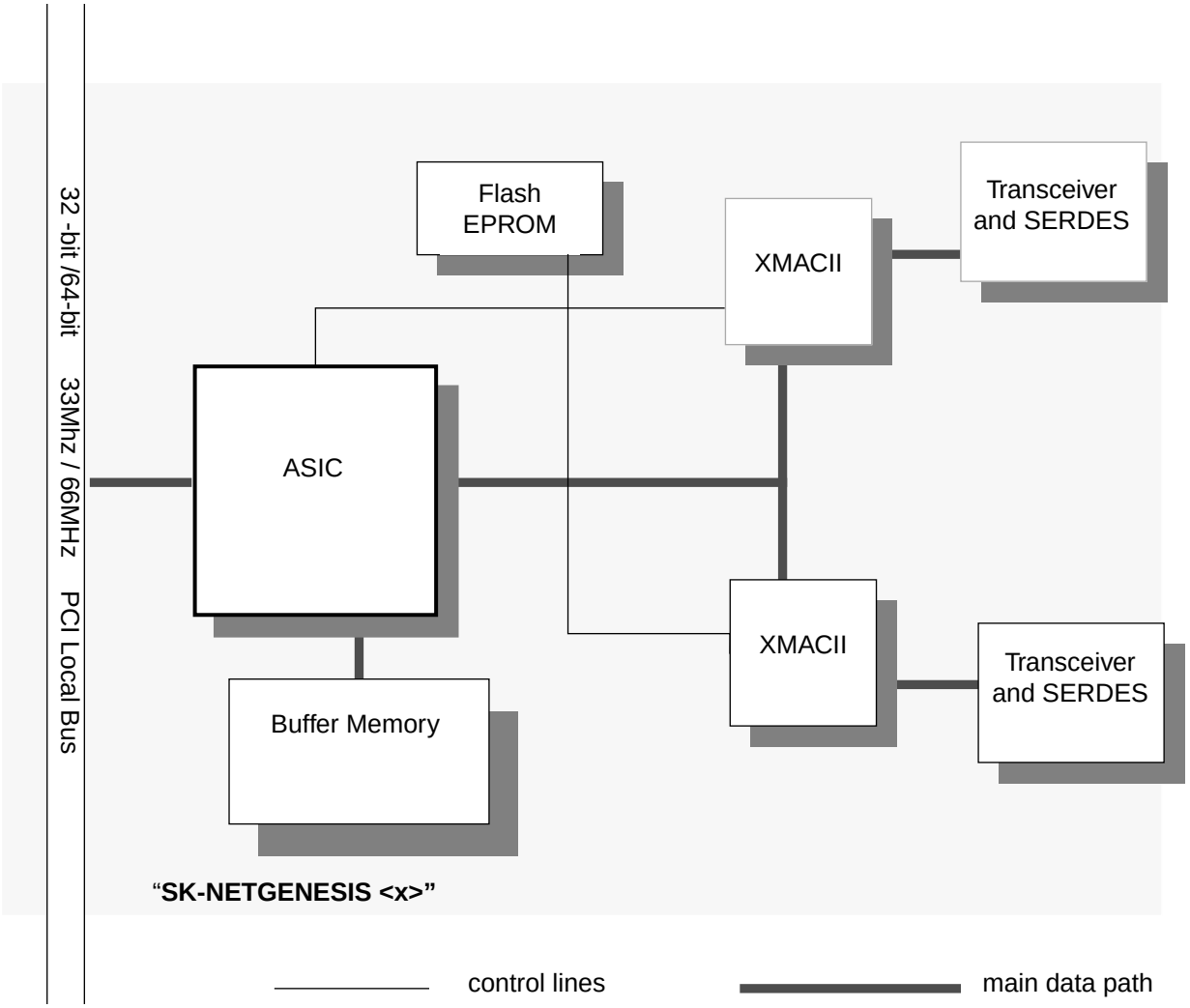
1.1.4 Network Interface Card

Below some characteristics of the "**SK-NET GENESIS <x>**" are listed that refer to the entire network interface card:

- ❏ Manufacturing options for single link and/or dual link.
  A two-MAC version of the "**SK-NET GENESIS <x>**" may be programmed to provide FDDI-like dual homing for redundancy and increased reliability: The NIC may be attached to two different switches while only one connection is active. With this a redundant data path is available.
- ❏ 1 MByte buffer memory for receive and transmit queues (manufacturing option for smaller and larger buffer memory)
  425 MB/s buffer memory bandwidth are provided to transfer frames to/from the network side and to/from the host system's PCI bus side simultaneously.
- ❏ PCB prepared for an industry standard 1x9 pinout package for the optical transceivers, e.g. 850nm or 1300nm with Integral Duplex SC Connectors
- ❏ Temperature sensor and voltage sensors

2.0 Physical Blocks

This survey describes the structure of the "**SK-NET GENESIS <x>**" card and the functions of essential card elements. The block diagram displays



the controller's main physical devices. Most of it must be programmed or accessed to operate the card.

- ❏ Reprogrammable Flash EPROM (FPROM)
- ❏ Application Specific Integrated Circuit (ASIC)
- ❏ Buffer memory
- ❏ XMACII Gigabit Ethernet controller
- ❏ Gigabit Ethernet transceiver and SERDES

The ASIC interfaces the PCI system bus on one side and the XMACII Gigabit Ethernet controller(s) on the other side. Most of the NIC's registers to control the hardware are implemented in the ASIC, except control and operation register of the XMACII medium access controllers. The Flash EPROM holds reset/start values that are loaded into the registers to bring the NIC in a clearly defined state after start-up or reset. It may also hold bootcode.
The "**SK-NET GENESIS <x>**" also supports the optional Vital Product Data (VPD) as proposed in the PCI specification. Following the specification the VPD are read out from a dedicated I^2C-attached EEPROM - they are not included in the Flash EPROM.

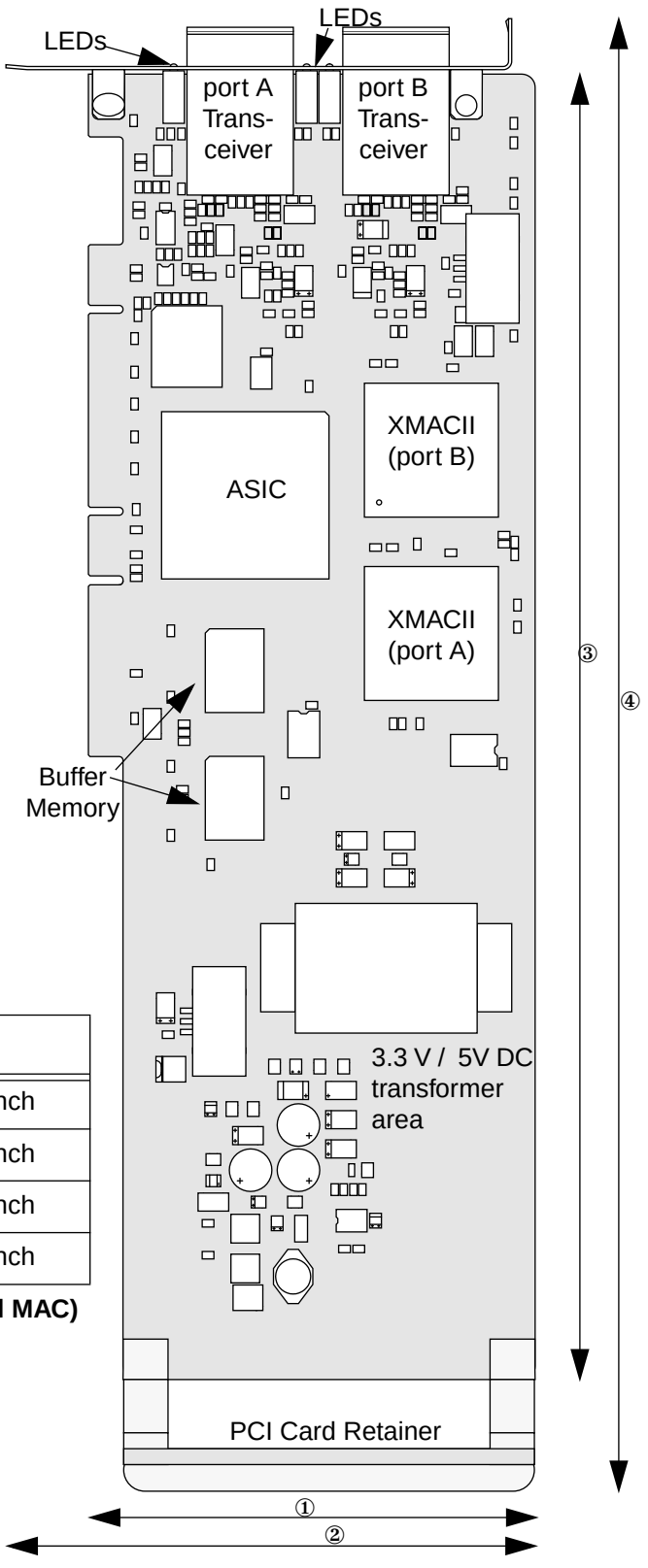
The main physical devices are involved in handling the payload data: the ASIC, the buffer memory, the XMAC, and the transceiver.

Some devices on the NIC are not listed above:

- ❏ The DC-transformer providing 3.3 V on-board power supply (the adapter is only connected to the 5V power line from the PCI bus - not to 12V, nor to the 3.3V from the PCI slot - however universal signalling, i.e. both 3.3V and 5V signals at the PCI slot are provided.)
- ❏ The temperature sensor the voltage sensors and the I^2C bus.
- ❏ EEPROM containing the PCI VPD (Vital Product Data) according to PCI 2.1 ECR (Engineering Change Request)

The following figures show the arrangement of devices on the printed circuit board for two versions of the NIC as an example.

2.1 Printed Circuit Board



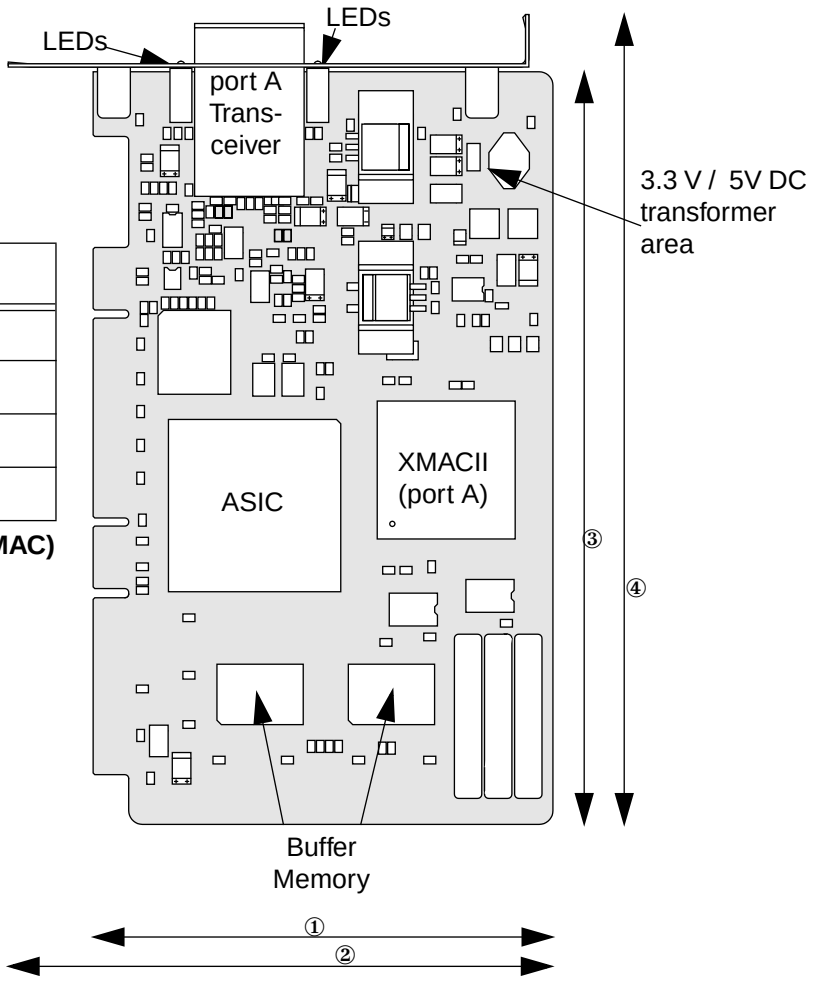
Dimensions:		
①	107 mm	4.20 inch
②	127 mm	5.00 inch
③	312 mm	12.3 inch
④	352 mm	13.7 inch

Table 1: Dimensions (Dual MAC)

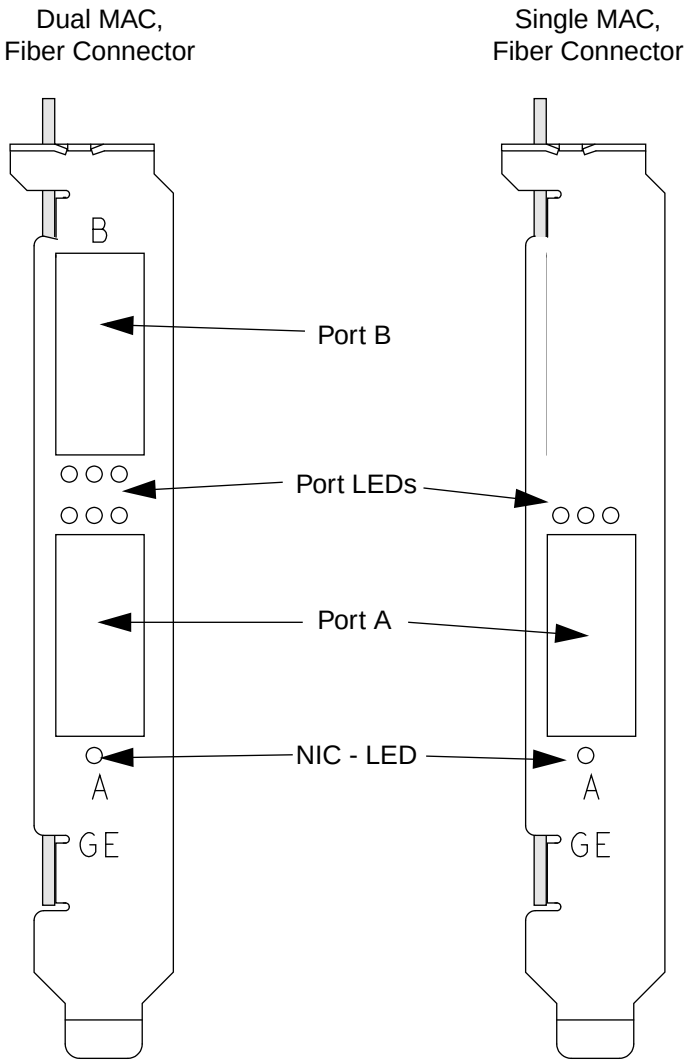
"**SK-NET GENESIS <x>**" Dual MAC

Dimensions:		
①	107 mm	4.20 inch
②	127 mm	5.00 inch
③	176 mm	6.9 inch
④	189 mm	7.4 inch

Table 2: Dimensions (Single MAC)



"**SK-NET GENESIS <x>**" single MAC



"**SK-NET GENESIS <x>**"

2.1.1 LEDs

The behavior of LEDs is software controlled - actual behavior is driver dependent. Therefore the following table can only show the design intention.

The "**SK-NET GENESIS <x>**" comes with one NIC LED and 3 port LEDs per MAC.
The NIC LED is yellow and is intended to indicate that a driver is started. (This does not mean that there's a connection established.) The port LEDs are intended for the following:

Color (from left to right)	Intended Use	
green	**Link State**	may be on / off / blinking or may forward the XMAC's **/LINK_SYNC** signal (the connection is switched via software setting)
yellow	**Receive**	**on** when receiving frames
yellow	**Transmit**	**off** when transmitting frames

Table 3: LEDs - Intended Usage

2.2 Functional Description

The following gives a short description of the block diagram's components. A more detailed discussion of the logical units most of them implemented in the ASIC is subject of the following chapters. For detailed information about the XMACII Gigabit Ethernet Controller you should refer to the manuals and documentation from XaQti Corporation. Complete information about XMACII beyond its external interfaces is out of the scope of this manual.

2.2.1 ASIC

The Application Specific Integrated Circuit (ASIC) directly interfaces the PCI Local Bus on one side and the Gigabit Ethernet controller on the other side. Besides this, it handles access to the memory buffer via an integrated MMU. The ASIC comprises the main control units of the network interface card. The initialization values for the registers implemented in the ASIC are hardwired or can be overwritten by initialization values from the Flash EPROM. The ASIC contains the main logical units.

Bus Interface Unit (BIU)

The Bus Interface Unit (BIU) provides the framework for interfacing the PCI Local Bus. It comprises several state machines running synchronously with the PCI bus CLK signal. Except the interrupt signal that is independent from the PCI bus CLK, all outputs are synchronously generated with the rising edge of CLK. All inputs are synchronously sampled. The BIU handles

- ❏ Access to the configuration data
- ❏ Access to I/O resources and/or memory mapped resources
- ❏ PCI bus master operation of the "**SK-NET GENESIS <x>**"

Buffer Management Unit (BMU)

The Buffer Management Units (BMU) are build upon the BIU services. At the PCI bus the ASIC provides 6 parallel data transfer queues, 2x2 transmit and 2x1 receive queues to optimize controlled data transfer between the card's buffer memory and the system RAM. Data is transferred by the "**SK-NET GENESIS <x>**" operating as PCI bus master.
The BMUs, one BMU per queue, control bus master data transfer be-

tween the buffer memory and the system RAM. For this, the BMU expects a preallocated and initiated buffer structure in the host system RAM. It's to the network driver to provide this structure for each BMU.

Data transfers over the PCI bus are 4 bytes or 8 bytes wide depending on the bus width (32 bit/64 bit). The transfer rate also depends on the bus clock (0-33 MHz / up to 66 MHz). The FIFO and buffer memory implementation of the "**SK-NET GENESIS <x>**" provides data rates and data space for PCI burst transfers with up to 512 bytes burst length without any need for the network interface card to insert wait states on the PCI bus.

Arbiters

PCI access to the 6 independent concurrent queues is controlled by the PCI arbiter according to a rotating priority scheme. Transfer length is optimized for system dependent cache line sizes and best PCI command usage (descriptors and data).

There are additional arbiters controlling

- ❏ Priority between transmit queues (rate control)
- ❏ Access at the buffer memory <-> PCI FIFO interfaces
- ❏ Access at the buffer memory <-> XMAC(s) interface

Each arbiter operates independently on a per request base. (Though there's usually no direct interaction between arbiters, for example arbitration between transmit queues, which can be used for rate control, can influence the traffic pattern by giving precedence to one queue.)

2.2.2 Buffer Memory

The "**SK-NET GENESIS <x>**" controller typically comes with a 1 MByte SRAM for frame buffering. The buffer memory is controlled by the ASIC. Various amounts of buffer memory were foreseen as hardware design option:

Memory Size	# RAM devices	RAM type
512 KBytes	2	64K x 36 bits
1 MByte	2	128K x 36 bits
2 MBytes	4	128K x 36 bits

Table 4: On-Board Memory Options

2.2.3 XMACII

The XMACII from XaQti Corporation controls access to the transmission medium. This implementation complies with the IEEE 802.3z specification. The XMACII mounted on the "**SK-NET GENESIS <x>**" provides interfaces to various transmission media transceivers e.g. fiber optic transceivers and 1000BASE-T transceivers.

The "**SK-NET GENESIS <x>**" comes in two versions with one XMACII assembled (single MAC version) or with two XMACs assembled (dual MAC version). On dual MAC versions of the "**SK-NET GENESIS <x>**" the two XMACs are connected to the ASIC via a common data bus. The MAC ar-

biter in the ASIC can switch the data bus between the XMACs. The common 32-bit wide data bus between ASIC and the XMAC(s) is operated at 53.12 MHz providing a total bandwidth of 1.7 Gbit/s.

2.2.4 Transceiver and SERDES

The SERializer and DESerializer (SERDES) and the transceiver are regarded as a logical unit throughout this manual. The SERDES translates the 10-bit parallel data stream to the 1 Gbit/s serial stream and vice versa. One SERDES per XMACII is mounted on the back side of the printed circuit board.
The transceivers transform the serial signals from the SERDES to the optical/electrical signals as defined in the corresponding PMD (Physical Media Dependent) specification.

The network interface card is designed to support different media specified in the IEEE specification framework For the first NICs this will be

- 1000 BASE-SX
  62.5 µm / 50 µm multimode fiber with shortwave lasers
- 1000 BASE-LX
  62.5 µm / 50 µm multimode fiber with longwave lasers
  10 µm singlemode fiber with longwave lasers

Later on the copper cabling may be supported, too:

- 1000 BASE-T
  UTP (Category 5) cabling with 100m cable length distance.

3.0 Programming Interface

The PCI specification provides three different information spaces and different methods to access and control a NIC via the PCI bus:

- ❏ Configuration Space (Hardware access methods are Configuration Read and Configuration Write cycles)
- ❏ The Memory Space (There are several hardware access Methods on the PCI bus e.g. Memory Read, Memory Read Multiple, ...)
- ❏ I/O Space (translates to I/O Read, I/O Read Write cycles on the PCI bus)

The configuration space is treated in detail in Appendix A of this manual. The details how the **"SK-NET GENESIS <x>"** has implemented configuration space access, listings of the PCI configuration registers, the usage of optional extensions by the **"SK-NET GENESIS <x>"** network interface card can be found there.
While it is not recommended for normal use, for test and diagnosis software the PCI configuration space of the **"SK-NET GENESIS <x>"** is mapped into the control register file that can be accessed via I/O and memory accesses, too.

3.1 I/O Space and Memory Space Basics

The "**SK-NET GENESIS <x>**" registers can be mapped in both I/O space and memory space. It depends on some initialization values in the configuration space whether the network interface card uses I/O mapping, memory mapping or both.
Depending on the BIOS settings/the operating system, the "**SK-NET GENESIS <x>**" is mapped into the I/O space only, mapped into the memory space only, or mapped into both. The PCI specification "highly recommends" mapping into the memory space over mapping into the I/O space. The "**SK-NET GENESIS <x>**" supports any of these mapping combinations. The NIC claims

- ❏ 256 Bytes of 32-bit I/O space
  and
- ❏ 16 KBytes of 32-bit memory space

A memory base address and an I/O base address are assigned to the NIC by the system software. The assigned base addresses are written to the configuration space header.

Thus, one can access the NIC's registers, e.g. control/status registers, timer, interrupt mask and source register, etc. either via I/O accesses or memory accesses.

All the registers of the "**SK-NET GENESIS <x>**" are comprised in the **Control Register File**. Additionally the configuration space registers are mapped in the control register file.

If the NIC is mapped into both, memory space and I/O space, the control register file can be accessed via I/O accesses and via memory accesses. With both types of access you operate on the same registers.

The control register file comprises all control information necessary to operate the NIC. For example some control functions:

- ❏ Control of the various state machines realized in the ASIC
- ❏ Control of data transfer between the host memory and the on-board buffer memory.
- ❏ Control of the XMACII
- ❏ Interrupt sources and interrupt mask
- ❏ On-board timer
- ❏ ... .
- ❏ All you need to operate the card

"Control of .. " the card's registers does not mean that driver software has to poll or access the registers continuously. The main task for drivers after initialization is interrupt handling. The great majority of registers must be accessed for initialization only or for control by diagnosis software.

3.1.1 I/O Mapping

The registers mapped in the I/O space have to be accessed with the minimum data width, i.e. 8-bit, 16-bit, or 32-bit transfers as the registers are defined.

The I/O space of the "**SK-NET GENESIS <x>**" is mapped in the host's I/O space. It spans 256 Bytes. To make all registers of the control register file available via I/O accesses a Register Address Port (RAP) is provided. The entire 16 KBytes address space of the control register file is divided in 128 × 128 Bytes blocks. Giving the block number and the address relative to the begin of a block <0x80 .. 0x00> enables to address any register in the 16 KBytes control register file.

Block 0 is permanently mapped to the lower half - the lower 128 Bytes out of 256 Bytes I/O addresses. The block specified by the data in the Register Address Port (RAP) is mapped to the upper half - the upper 128 Bytes. The RAP is the first byte at the relative address 0x00 in block 0 of the control register file. Thus writing to the I/O base address specifies the RAP value.

Bit	Name	Description	Write	Read	Reset (SW)
31..8	Reserved		ne	0	
7..0	RAP	Specifies one out of block 0 ..127, which is mapped to the upper half of the 256-Byte I/O range. 0=block 0, ...0x7f=block 127	yes	aw	0

Table 5: Register Address Port (RAP)

To address any register in the Control Register File via I/O access, write the block number to the I/O base address and access

<Register Address> = <I/O base address> + 0x80 +
<register address relative to the begin of the block>

(Adding 0x80 means switching to the upper half of the 256 Byte I/O space).

3.1.2 Memory

There are two accessible resources:

- ❏ Memory mapped registers (Control Register File)
- ❏ Expansion ROM (Flash EPROM)

The registers mapped in the memory space have to be accessed with the minimum data width, i.e. 8-bit, 16-bit, or 32-bit transfers as the registers are defined. The Expansion ROM can be accessed with 8-bit, 16-bit, or 32-bit transfers.

3.2 Bus Interface Unit (BIU)

The Bus Interface Unit (BIU) handles the basic protocol for accesses via the PCI bus. It is built of several state machines running synchronously to the PCI bus CLK signal. All inputs are synchronously sampled. All outputs are synchronously generated on the rising edge of CLK.

The target section of the network interface card is a 32-bit device. It is mapped in the lower 4GB of the address space and therefore ignores all Dual Address Cycles.

The master section of the network interface card is capable of addressing the whole 64-bit address space independent of the slot size it is mounted in. It uses Single Address Cycles for accesses to addresses below 4GB and Dual Address Cycles for accesses above 4GB. It uses 64 bits for data transfers to and from 64-bit targets, if mounted in a 64-bit slot, and 32 bits for data transfers to and from 32-bit targets or if mounted in a 32-bit slot.

The master section of the BIU will be treated in more detail when data transfer over the PCI bus is discussed. For the actual programming interface it is not of interest since bus master operations are performed by the "**SK-NET GENESIS <x>**" hardware according to the preconfigured control registers.

3.2.1 Slave Access to Configuration Space

Slave access to the configuration space is also not of primary interest for the programming interface. However it may be valuable for troubleshooting with PCI bus analyzers and exerciser tools. This bus operation is performed by the target sequencer state machine. Acceptance and termination of a transaction are determined by a backend consisting of the configuration decoder and the configuration register file.

The network interface card is responding to Type 0 configuration accesses (**AD<1..0>** = "00", **IDSEL#**). If the configuration space is targeted for a burst operation, it responds with a disconnect with the first data transfer.

The configuration register file can be accessed with 8-bit, 16-bit or 32-bit transfers.

Configuration transactions are not aborted (Target Initiated Termination). On read transactions all data is driven as defined for full 32-bit accesses independent of BE<3..0>#.

3.2.2 Slave Access to I/O-Resources

This bus operation is performed by the target sequencer state machine. Acceptance and termination of a transaction are determined by a backend consisting of the control decoder and the control register file.

The control registers have to be accessed with the minimum data width (8-bit, 16-bit or 32-bit) transfers as the registers are defined. If the I/O resources are targeted for a burst operation, it responds with a disconnect with the first data transfer.

On read transactions all data is driven as defined for full 32-bit accesses independent of BE<3..0>#.

3.2.3 Slave Access to Memory Resources

There are two accessible resources:

- ❏ Memory Mapped I/O-Resources
- ❏ Expansion ROM (Flash-EPROM)

Accesses to memory mapped I/O-Resources are performed by the target sequencer state machine. Acceptance and termination of a transaction are determined by a backend consisting of the control decoder and the control register file.

The control registers have to be accessed with the minimum data width (8-bit, 16-bit or 32-bit) transfers as the registers are defined.

On read transactions all data is driven as defined for full 32-bit accesses independent of BE<3..0>#. If the memory resources are targeted for a burst operation, it responds with a disconnect with the first data transfer.

Expansion ROM

Accesses to Expansion RROM are performed by the target sequencer state machine. Acceptance and termination of a transaction are determined by a backend consisting of the Flash EPROM Decoder and the Flash EPROM translation machine.

The Expansion ROM can be accessed with 8-bit, 16-bit or 32-bit transfers. On read transactions all data is driven as defined for full 32-bit accesses independent of BE<3..0>#.

Both cases:

All PCI memory cycles are defaulted to simple memory read and memory write cycles.

3.2.4 Host Clock Related Resources

The "**SK-NET GENESIS <x>**" operates on several clock domains. For example write access to the register control file come synchronous with the PCI clock. On the network interface card they may cross the domain border and operate on registers that are run synchronously with the host clock (~ the NIC's operating clock). Crossing the domain border causes some delay for the PCI related target state machine to complete the access cycle on the PCI bus. Thus, all resources that are clocked with host clock are doing write posting to avoid violation of the 16 clocks rule for accesses to all PCI targets.

Each resource, that is integrated in the ASIC (timer, IRQ moderation timer etc.), has its own set of posting registers, that hold subaddress, byte enables and write data.

Also each external device (XMACII and optional external registers) has its own set of posting registers, but they are sharing the board's internal address and data bus.

While a posted write is in progress, all following accesses to this resource are terminated with a target retry until the posted write is completed. Accesses to resources sharing the same set of posting registers are terminated with a target retry, while a posted write through this set of registers is in progress. Accesses to resources, that don't share the busy set of posting registers are completed normally on the PCI bus without additional wait states.

The Flash EPROM doesn't implement write posting and delayed read transactions. For accesses to the Flash EPROM the posting registers are set to transparent mode. While an access to the Flash EPROM is in progress all accesses to the other resources sharing the internal address and data bus are terminated with a target retry. Accesses to the Flash EPROM are terminated with a target retry, if an access to another resource sharing the internal address and data bus is in progress.

Exception: While a posted write or a delayed read operation is in progress, all accesses to the Register Address Port (RAP) are terminated with a target retry to assure RAP consistency.

4.0 Data Path - Overview and Details

The **"SK-NET GENESIS <x>"** Network Interface Card (NIC) reads data from the host system's RAM and puts it on the network and vice versa. For achieving high performance with minimum load on the system CPU(s) the main steps of this task are performed by the NIC hardware.
Once initiated, the network driver has to handle network data in the system RAM and to respond to interrupts from the network interface card. The driver may initiate transfers and control the hardware. However, time consuming or CPU intensive operations like moving the data to/from the system RAM are handled by the **"SK-NET GENESIS <x>"** hardware. Interrupt moderation and aggregation of data on the transmit data path may additionally reduce the load on the system CPU.

As seen from the NIC data transfer is splitted in two tasks:

- ❏ Moving data between the system memory and the NIC - this is done by the ASIC.
- ❏ Putting data on the network / read data from the network - this is done by the XMACII Gigabit Ethernet controller and the ASIC.

The on-board buffer memory, the FIFOs in the ASIC and in the XMACII provide some interim storage for frames. The FIFOs first task is to enable data flow -even for PCI burst transfers- across independently clocked devices and to provide a contiguous data stream over the clock domain borders. Another task of the FIFOs is data band width adaptation: The MAC FIFO for example translates from the 64-bit (+parity) data path within the ASIC to the 32-bit wide data port on the XMACII. The on-board buffer memory is for interim storage: congestion avoidance, preventing receive overflows, or for TCP/IP checksum insertion.

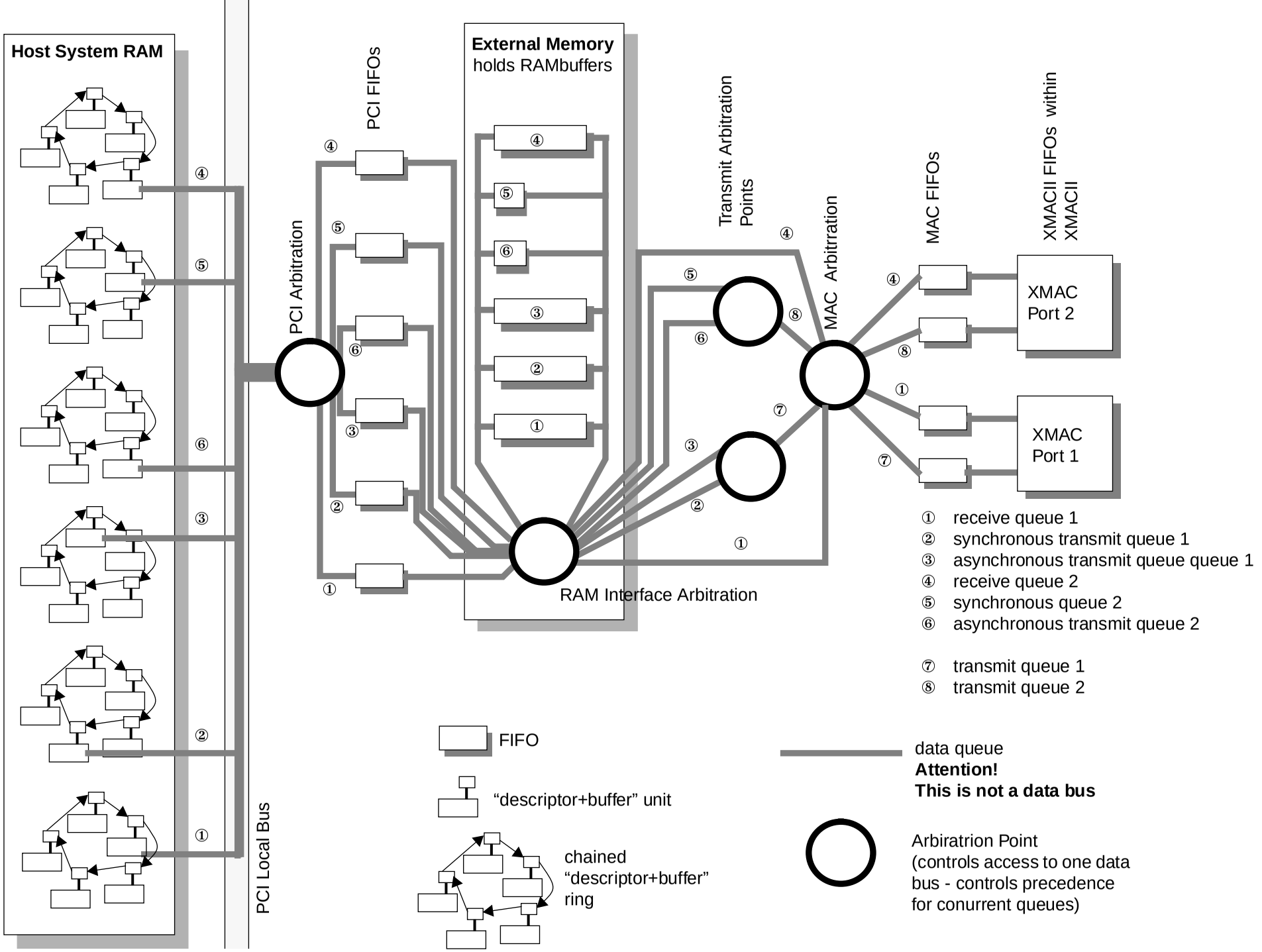
Operation and functions of the XMACII are described in detail in the XMACII datasheets from XaQti Corporation. Thus this manual will mainly refer to operation and control of the ASIC.

Accesses to control the **"SK-NET GENESIS <x>"** is handled by the "target state machine". As seen from the PCI bus the NIC is the target of accesses. E.g. the driver sets register values or clears interrupts on the NIC. Target accesses are 32-bit accesses. The **"SK-NET GENESIS <x>"** is operated as CPU slave. Transferring payload data in the slave mode is possible for test purposes.

However, to achieve maximum performance at minimum CPU load data transfer between the **"SK-NET GENESIS <x>"** and the system memory is done by the NIC operating as PCI bus master. Depending on the PCI slot data may be transferred over the PCI bus with 32-bit or 64-bit data width.

4.1 Receive Queues - Transmit Queues

Data is moved between the system RAM and the network by queueing them on a data path - a receive or transmit queue. Receive queues are for data frames coming from the Ethernet. Transmit queues are for putting data from the host system onto the Ethernet network. The **"SK-NET GENESIS <x>"** comes with a total of 6 data path or queues at the PCI side and 2 data transfer queues at each XMAC.
For each of the 6 queues at the PCI side the driver software has to provide a "descriptor+buffer" chain that holds data frames or frame fragments together with control information that determines how the **"SK-NET GENESIS <x>"** hardware handles the frame and frame-related status in-


Host System RAM
PCI Local Bus
PCI Arbitration
PCI FIFOs
External Memory
holds RAMbuffers
RAM Interface Arbitration
Transmit Arbitration
Points
MAC Arbitration
MAC FIFOs
XMACII FIFOs within
XMACII
XMAC
Port 2
XMAC
Port 1
① receive queue 1
② synchronous transmit queue 1
③ asynchronous transmit queue queue 1
④ receive queue 2
⑤ synchronous queue 2
⑥ asynchronous transmit queue 2
⑦ transmit queue 1
⑧ transmit queue 2
FIFO
“descriptor+buffer” unit
chained
“descriptor+buffer”
ring
data queue
Attention!
This is not a data bus
Arbiratrion Point
(controls access to one data
bus - controls precedence
for conurrent queues)

formation.
The design idea is that the network driver provides several descriptor+buffer units per queue, a chain of descriptor and assigned buffers for each queue. Each descriptor+buffer unit contains information where to read/write data in the system RAM and where the BMU (Buffer Management Unit) will find the descriptor structure of the next unit. The driver software manages the data buffer and descriptor chain in the system memory while the BMUs read/write the units and transfers the data buffers according to the information encoded in the respective descriptor. The BMUs handle transfer along the PCI bus.

A descriptor+buffer unit provides a data container for one frame or for a fragment of a frame. Or vice versa each frame requires at least one descriptor+buffer unit for transfer between the NIC and the system RAM. The STart of Frame (STF) flag and the End Of Frame (EOF) flag in the descriptor identify frame borders.
The driver software provides the buffers and descriptor structures in the system RAM and may initiate the BMUs to perform data transfers. Once initiated the BMUs transfer a frame or subsequent frames over the PCI bus at maximum transfer rate without CPU load penalty. Data flow is controlled only through descriptors (except for error handling). All FIFOs (including the BMUs) are intended to run without any software interaction after initialization.

General Remarks

As shown in the figure there are lots of FIFOs involved in data transfer. Each FIFO has a write state machine and a read state machine running as independently as possible. A transfer at STart of Frame (STF) is initiated by the data source. Data transfer is requested, if the source has data available and there's space left at the destination. End Of Frame (EOF) is tracked from the source and forwarded to the destination. All transfers are limited by timeout counters.

The FIFOs, except the PCI FIFOs are accepting more than one packet. The PCI FIFOs are optimized that burst transfers are not broken at packet boundaries, but may be suspended for some clock cycles. (Under worst case load conditions, the system is driven to max length bursts for each participant.)

4.1.1 Frame Metadata - Internal Status

While in the host system's RAM the descriptor offers accompanying control/status information for each frame, there's a similar concept when payload data is moved along their queues: the **Internal Status**. The internal status comprises up to 4 64-bit words that accompany the payload data over the network interface card carrying the additional metadata about the payload and control information for each frame separately.

The descriptor structure and contents will be shown in detail when starting a closer look at a frame's path from the host system to the network. Descriptors are clearly located, read and written in the system RAM, especially from a driver point of view. The internal status however is a concept that will be mentioned at various points of a frame's data path over the network interface card. (The internal status is mostly invisible for

driver / test software it can only be read from the RAMbuffers. Nevertheless it's important for arbitration decisions since it contains frame border flags.)

Internal Status

On the network interface card the first two 64-bit words of the internal status are inserted in the datastream. (They are removed before giving data to the XMAC.) The 3rd and 4th 64-bit word of the internal status are defined for frames on the transmit queues, i.e. frames send to the network, only. They are used for control purposes.

The internal status words are passed along several stations on the network interface card together with the payload data. Stations may evaluate/set/modify the internal status. The complete internal status is validated by a flag **Internal Status Valid** by the creating/modifying state machine. The **Internal Status Valid** flag is cleared by the destination.

Receive Queues:

	Internal Status Register/Receive (64-bit word 1 & 2)		Valid at STF	Valid at EOF	Default
Internal Status Word 1 (lower address when read from RAMbuffer)					
63:52	**Reserved**				
51:32	**Addr Frame End**	Internal address of the frame end **Addr Frame End** is individual for each FIFO and set to ZERO, if forwarded to other FIFOs		✓	
31:16	**Frame Length**	Frame length in bytes (defined by the XMACII interface)		✓	
15:2	**Reserved**				
1	**Time Stamp Valid**	Set, if timestamp appended by XMAC		✓	
0	**RX Status Valid**	Set, if Receive Status appended by XMACII (Receive Overflow)		✓	
Internal Status word 2 (upper word when read from RAMbuffer)					
63:32	**Time Stamp**	Timestamp as defined by XMAC. Set to ZERO, if not appended by XMAC		✓	
31:0	**RX Status**	Receive status as defined by XMAC. Set to ZERO, if not appended by XMACII (Receive Overflow)		✓	

Table 6: Internal Status - Receive Queues

Transmit Queues

	Internal Status Register/Transmit (64-bit word 1 & 2)		Valid at STF	Valid at EOF	Default
Internal Status word 1 (lower word when read from RAMbuffer)					
63:52	**Reserved**				
51:32	**Addr Frame End**	Internal address of the frame end **Addr Frame End** is individual for each FIFO and set to ZERO, if forwarded to other FIFOs		✓	
31:16	**Frame Length**	Frame length in bytes (defined by BMU)		✓	
15:3	**Reserved**				
2	**Insert Check-sum**	**Insert Checksum** as defined by the transmit descriptor with **STF** set	✓		
1	**Dis CRC**	**Dis CRC** as defined by the transmit descriptor	✓		
0	**Reserved**				
Internal Status word 2 (upper word when read from RAMbuffer)					
63:32	**Reserved**				
31:0	**TX Status**	Transmit Status as defined by transmit descriptor.		✓	

Table 7: Internal Status - Transmit Queues

	Internal Status Register/Transmit (Qword 3 & 4)		Valid at STF	Valid at EOF	default
	Internal Status Word 3				
63:17	**Reserved**				
16	**St&Fwd**	Store and Forward Prevents to start reading from RAMbuffer before complete frame is available (including written checksum and status)	✓		

Table 8: Internal Status Word 3 & 4 - Transmit Queues

15:0	**TCP Sum Write**	**TCP Sum Write** as defined by the transmit descriptor.		✓	
	Internal Status Word 4				
63:0	**Check Sum**	Checksum packed in the 64-bit, which has to be replaced		✓	

Table 8: Internal Status Word 3 & 4 - Transmit Queues

STF, EOF

STart of Frame (STF) and **End Of Frame** (EOF) flags indicate frame boundaries. They are used by several arbiters and state machines. The STF and EOF bits are required since a frame may be fragmented into several non-contiguous buffers in the host memory. To transmit this splitted frame each fragment requires its own descriptor. Additionally, the end of the frame transfer can be indicated by an interrupt.

4.1.2 TCP/IP Checksum Calculation

The TCP/IP checksum calculation is a feature of the "**SK-NET GENESIS <x>**" to reduce CPU load by handling the most part of it in the NIC's hardware. Depending on the descriptor's **OPCode** settings it can be used or disabled. It also depends on the software, especially the network protocol stack implementation, whether and how effectively this feature reduces CPU load and improves the overall performance.

The TCP/IP hardware checksum is a 16-bit one's complement checksum as defined in RFC 793 "Transmission Control Protocol". The "**SK-NET GENESIS <x>**" hardware calculates checksums along an Ethernet packet starting at an offset defined in the receive/transmit descriptor until **End Of Frame**[1]. The result of the checksum calculation belongs to a frame's metadata and is contained in the descriptors and internal status words respectively.

On the receive queues the hardware writes the calculated checksum into the last descriptor that belongs to a frame, i.e. the **End Of Frame** is set to "1" in the descriptor. The driver software can use the hardware-calculated checksum to compare it with the checksum included in the frame for frame validity checks.
On the transmit queues the hardware receives a frame from the system RAM, writes the entire frame into the on-board buffer memory (**St&Fwd On** is set). Then the calculated checksum is added to an offset and written over the frame at the checksum write position. The offset and the checksum write position are defined in the first transmit descriptor of the frame. Addition of offset and calculated checksum is done according to one's complement arithmetic.

On transmit queues the "**SK-NET GENESIS <x>**" computes one checksum starting at a frame position defined in the frame's first descriptor. On receive queues always two checksums are computed. The receive de-

1. End Of Frame, Start Of Frame flags of the "**SK-NET GENESIS <x>**" refer to Ethernet (MAC) frames not to IP, TCP, UDP or other protocol frames.

scriptor holds the start position for checksum computing. It's to the driver software to set appropriate start positions when initializing the receive descriptors.

Example:

The TCP/IP checksum calculation can be used to calculate the TCP/IP checksum over the most part of a TCP/IP or UDP/IP frame. E.g. when transmitting an entire TCP/IP frame within an Ethernet packet, driver software can calculate the IP checksum which is a checksum over the IP header only and set the transmit descriptor to calculate the checksum over the IP data (= TCP frame). (Driver software computes the checksum of the Pseudo Header - not the hardware.)
For the receive queues the TCP/IP checksum calculation can be used to calculate e.g. one checksum starting with the IP header until **EOF** the second checksum starting with IP data (= TCP frame) until **EOF**. Since the "real" IP header checksum refers to the IP header only, the IP data checksum must be subtracted from the first calculated checksum appropriately. Depending on the driver interface and the implementation of the TCP/IP protocol stack, the hardware calculated checksum for received frames can also be used for IP frames that exceed the Ethernet frame size and are received within several Ethernet frames. However, checksum calculation over several Ethernet frames is not possible for the transmit queues.

☞ **The checksum calculated by the hardware needs some modification before or after sending or receiving a packet to or from the network, respectively. Thus, it is important to understand the Internet checksum algorithm in detail and how to implement the code to modify the checksum.[1]**

4.2 System RAM - Descriptors and Buffers

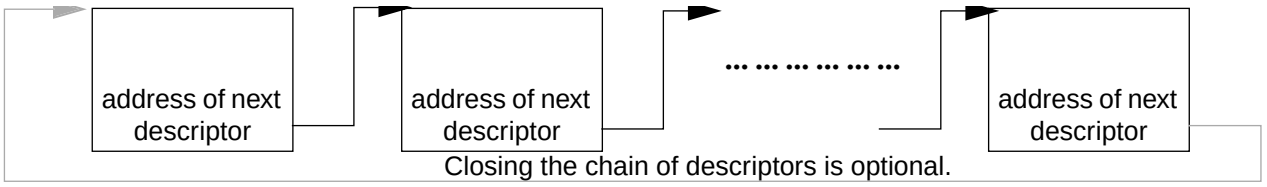
The descriptors provide address information for the BMUs. The descriptor holds an address for an assigned data buffer, a container for the data frame or frame fragments to be sent/received.
Besides this descriptors also carry control and status information, e.g. the frame status word and timestamps for further processing by the network driver. When sending frames, i.e. for the transmit queues, the descriptor enables/disables the hardware TCP checksum computing on the "**SK-NET GENESIS <x>**" network interface card.

Usually the driver software provides a chain of descriptor+buffer units at driver initialization time: one chain per queue. Descriptors of transmit queues are different from the descriptors of the receive queues. Receive queue descriptors contain status information about the received Ethernet frame while the transmit descriptors provide information for the hardware how to handle the frame - frame metadata. Thus, for each of the six queues a chain of descriptors should be build during driver initialization. It's also possible (not necessary) to chain the descriptors in a descriptor ring as sketched in the figure.

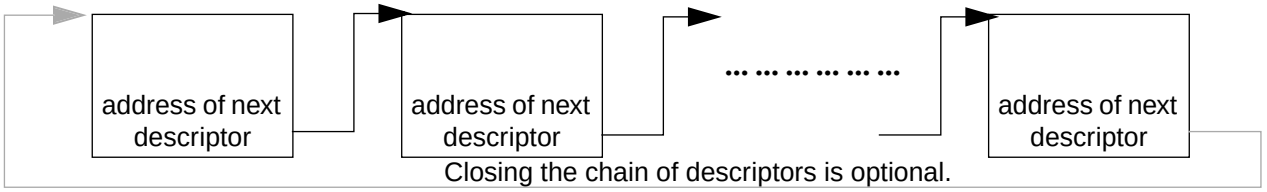
---

1. The following RFCs refer to TCP/IP checksum computing:
   RFC 791 "Internet Protocol"
   RFC 793 "Transmission Control Protocol"
   RFC 1071 "Computing the Internet Checksum"
   RFC 1624 "Computation of the Internet Checksum via Incremental Update"

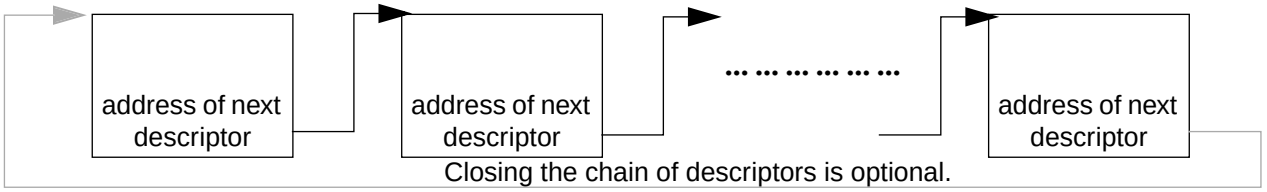
Chained Descriptors for Receive Queue 1



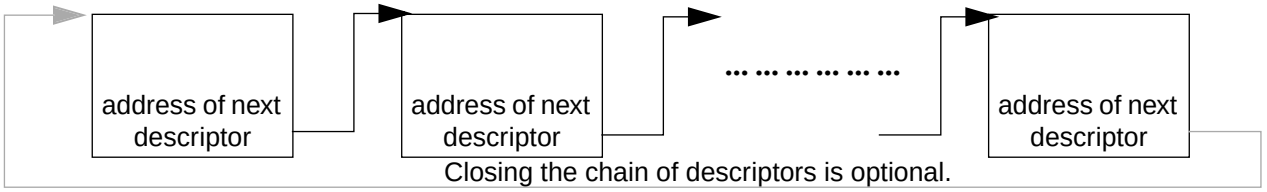
Chained Descriptors for Asynchronous Transmit Queue 1



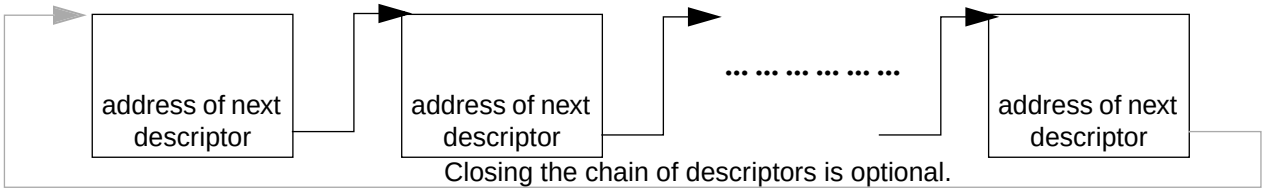
Chained Descriptors for Synchronous Transmit Queue 1



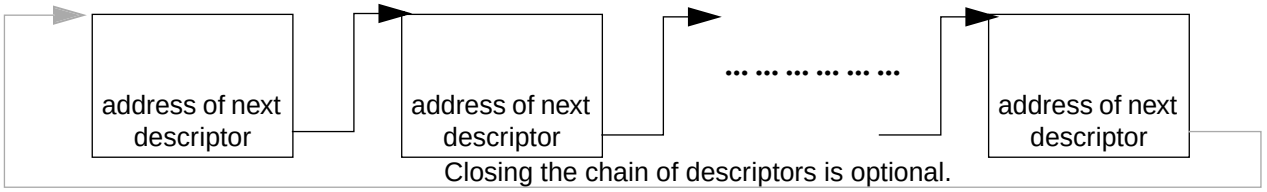
Chained Descriptors for Receive Queue 2



Chained Descriptors for Asynchronous Transmit Queue 2



Chained Descriptors for Synchronous Transmit Queue 2



Descriptor Chains

The number of descriptors for a queue is theoretically unlimited. The size of a descriptor is 8 × 32-bits.

☞ **The descriptors have to be located on 8-byte boundaries i.e. fitting for 64-bit accesses by the BMU (even if the NIC is installed in a 32-bit slot).**

All descriptors of a queue must be within a 4 GBytes address range (the address is specified in the descriptor by the lower 32-bit in 64-bit address space). The upper 32-bit in the 64-bit address space that specify the 4 GBytes range are configured once for each queue at driver initialization. The buffer addresses however can start at any byte address. The buffer address, a full 64-bit address, is defined for each buffer in the defining descriptor separately. I.e. the buffers containing the payload data can spread over the entire 64-bit address space. A hardware multiplexer cares that data frames on the network interface card are contiguous without any gaps or padding bytes due to byte alignment. The multiplexer can be accessed, read out for test and diagnosis purposes.

The hardware details: buffer management is implemented for each queue separately, i.e. there are six independent BMUs. The BMUs are structured similar. The BMUs for the receive queues are identical in detail and the BMUs for the transmit queues are identical in detail.

4.2.1 Little Endian - Big Endian

Data and descriptors moved from/to the receive/transmit queues are expected to be little endian ordered.

The byte order in the transferred data (64-bit) words is assumed to be little endian by the host bridge in big and little endian systems. The data path for the receive/transmit descriptor words (32-bit) may be switched to big endian using **Rev_Bytes_Desc** setting in the control register file.

Remark: In the known systems it is not necessary to change the byte order for the data or the descriptors, except for AIX systems, which may gain performance, if the descriptors are byte-reordered using **Rev_Bytes_Desc**.

The table below shows the effect of **Rev_Bytes_Desc** where byte 0 is the least significant byte, byte 7 is the most significant byte of a 64-bit word.

Rev_Byte_Desc	64-bit Word Byte Lane 7-6-5-4-3-2-1-0	32-bit 1st 32-bit Word Byte Lane 3-2-1-0	32-bit 2nd 32-bit Word Byte Lane 3-2-1-0	Application
0	7-6-5-4-3-2-1-0	3-2-1-0	7-6-5-4	little endian System
1	4-5-6-7-0-1-2-3	0-1-2-3	4-5-6-7	big endian System with little endian adjustment for data: read just descriptor words

Table 9: Little Endian vs. Big Endian

4.2.2 Own Bit

For each queue at least one descriptor structure is required in the host memory. The descriptor structure provides at least the following informations to the BMU:

- ❏ An Own bit indicating whether the host (driver software) or the BMU (the NIC hardware) owns the current descriptor
- ❏ Buffer address (address of data to be transferred)
- ❏ Buffer length (number of bytes to be transferred)
- ❏ Address of next descriptor in the chain.

Since a descriptor is read/written from two instances, the driver/diagnosis software and the BMU hardware (Buffer Management Unit), some rules are needed to avoid conflicts. Access is controlled via the descriptor's **Own** bit. The **Own** bit indicates ownership i.e. the right to change the descriptor contents.

Value	Meaning
0	Host is owner of the descriptor After writing/updating the descriptor the host relinquishes the descriptor to the BMU by setting the **Own** bit to "1".
1	BMU is owner of the descriptor After writing/updating the descriptor the BMU relinquishes the descriptor to the host by setting the **Own** bit to "0".

Table 10: Descriptor's Own bit

After building and initializing the descriptor chains, receive queue descriptors should be owned by the BMU, transmit queue descriptors should be owned by the host. The BMU Control/Status registers of each queue should contain the address of a descriptor, the "current descriptor".
Later on the BMU reads descriptor addresses and updates the information in the control register file - together with additional information and control bits to handle the BMU mechanism.

Other than the **Own** bit, some bits of the descriptor are set only by the host (while he is the owner) some bits are set only by the BMU owning the descriptor.

Usually[1] the BMU is not polling descriptors for ownership. A descriptor read is initiated by the driver/diagnosis software setting the command **Start xxx** for the transmit queues or if an additional descriptor for receive

1. As this sentence says software should initiate the data transfer process. However as seen from previous similar BMUs a Start xxx command may get "lost" in some environments due to CPU <-> cache related interim storage problems under certain conditions. The event was observed to happen on a several minute time scale. Thus the polling mechanism should be used to send a Start xxx to initiate transfer of ready descriptors regularly.

data is needed. The BMU is reading the descriptor entry of a queue pointed to by the queue's **Current Descriptor Address**, if the host is setting the command **START xxx** for the related queue.

It is supposed, but not mandatory, that the host is building the structures once at initialization time. The **Descriptor Base Address Register High** has to be loaded with the upper 32 bits of the 4GB block the descriptors are located and the **Current Descriptor Address** registers of each queue has to be loaded with the 32-bit offset address of the first descriptor entry of a queue within this 4GB block, before a command **START xxx** is set.

The next descriptor entry is read by the BMU, if the ownership of the current descriptor entry is relinquished to the host. If the descriptor entry is not owned by the network interface card, no further reading of this descriptor is performed. Rereading is initiated by the host setting the command **START xxx**.

The **START xxx** and the complementary **STOP xxx** command act upon the "supervisor state machine" which is one state machine in the BMU's state machine set. **START xxx** or the end of handling a descriptor causes the supervisor state machine to request the next descriptor read. It's to the driver/diagnosis software to provide enough receive descriptor+buffer units to avoid congestion on the receive paths. Received frames will be queued in the buffer memory. Later on the XMACII may send Pause frames as defined in the Gigabit Ethernet specification to its neighbor port.

4.2.3 Receive Descriptor

The receive descriptors have the same structure for each of the two receive queues

Bit	Name	Description	Address	Defined by
RBCTRL Receive Buffer Control				
31	**Own**	**Own** indicates that the descriptor is owned by the host (**Own** = 0) or the **network interface card** (**Own** = 1). The host sets **Own** for relinquishing the buffer. The BMU clears **Own** after filling the buffer pointed to by the descriptor entry and completion of its descriptor entries. Once the BMU or the host has relinquished the ownership of a descriptor, it must not change any field in the descriptor.	base	host network interface card
30	**STF** (Start of FRAME)	**Start of Frame** is coming with two functions: - Defined by the host: If set, the BMU is allowed to use this descriptor for the start of a received frame**.** If a descriptor for the start of a received frame is needed, subsequent descriptors (owned by the BMU) are read by the BMU until finding one **STF** having set. Released descriptors are written back with correct information (e.g. **OWN** is cleared). - Defined by the BMU: **Start of Frame** is set, if the descriptor is holding the start of a received frame. Valid, if **Own** is cleared by the BMU. True, if = 1		host network interface card
29	**EOF** (End of Frame)	**End of Frame** indicates that this is the last descriptor used by the BMU for this frame. If **En IRQ EOF** and **EOF** are set, an interrupt **IRQ EOF** is generated after relinquishing this descriptor. Valid, if **Own** is cleared by the BMU. True, if = 1		network interface card

Table 11: Receive Descriptor

Bit	Name	Description	Address	Defined by
28	**En IRQ EOB** (Enable Interrupt on End Of Buffer)	If **En IRQ EOB** is set, an interrupt **IRQ EOB Rx** is generated after relinquishing this descriptor. True, if = 1		host
27	**En IRQ EOF** (Enable Interrupt on End Of Frame)	Enable **IRQ EOF**, see **EOF**. True, if = 1		host
26	**Dev 0**	Don't transfer to system memory. True, if = 1		host
25	**Status Valid**	Indicates if the **Receive Frame Status Word** (one of the following 32-bit words) is valid. **RFSW** is not valid, if this packet is shortened by a receive overflow. **Status Valid** is valid, if **Own** is cleared by the BMU and **EOF** is set. True, if = 1 A receive frame mast be treated as corrupted, if one of both **Status Valid** or **Time Stamp Valid** are not set.		network interface card
24	**Time Stamp Valid**	Receive Frame timestamp valid. **Time Stamp** is not valid, if timestamp is not appended by XMAC Valid, if **Own** is cleared by the BMU and **EOF** is set. True, if = 1 A receive frame mast be treated as corrupted, if one of both **Status Valid** or **Time Stamp Valid** are not set.		network interface card

Table 11: Receive Descriptor

Bit	Name	Description	Address	Defined by
23:16	**CHECK/Opcode**	If the BMU is reading a descriptor owned by the BMU with an invalid **CHECK/Opcode**, an interrupt **IRQ CHECK Rx** is generated and no further descriptors are read (Supervisor state machine defaults to Idle state). Opcodes: 0x55: Default descriptor 0x56: Descriptor with TCP/IP checksum extension		host
15:0	**RBBC** (Receive Buffer Byte Count)	Receive Buffer Byte Count The host is defining the length of the buffer in bytes. The network interface card is rewriting this counter with the real number of bytes written to the buffer (valid, if **Own** is cleared) In some cases of error handling (e.g. receive overflow of XMAC's FIFO), **RBBC** may come back set to **Zero**. In such cases, **Status Valid** and/or **Time Stamp Valid** are not set furthermore.		host, network interface card
31:0	**NRDADR**	Next Receive Descriptor Address, lower 32-bit Receive descriptors must be 8-byte aligned, i.e. fitting for 64-bit reads by the BMU	base + 0x4	host

Table 11: Receive Descriptor

Bit	Name	Description	Address	Defined by
		(The upper 32-bit of the descriptor address are set once at initialization time for each queue in the control register file i.e. receive descriptors are restricted to be in a joint 4 GByte address space, which means no severe restriction since the 4 GByte descriptor address space can be anywhere in the 64-bit address space. Receive buffers however can be addressed anywhere in the 64-bit address space on a per descriptor base. Both the lower 32 bit and the upper 32-bit address is specified in the descriptor as shown below. I.e. data can be distributed over the entire 64-bit address space wherever the driver wants to have them.)		
31:0	**RBADR LO**	Receive Buffer Address, Lower 32-bit	base + 0x8	host
31:0	**RBADR HI**	Receive Buffer Address, Upper 32-bit	base + 0xc	host
31:0	**RFSW**	Receive Frame Status Word as defined by XMACII (including length). Valid, if **Own** is cleared by the BMU and **EOF (End of Frame)** is set. If not valid, the BMU is writing back, what it got at that place.	base + 0x10	network interface card
31:0	**Time Stamp**	Receive timestamp as defined by XMAC, if XMACII is configured for appending timestamp. If not, Time Stamp = 0. Valid, if **Own** is cleared by the BMU and **EOF (End of Frame)** is set. If not valid, the BMU is writing back, what it got at that place.	base + 0x14	network interface card
TCP/IP checksum extension				

Table 11: Receive Descriptor

Bit	Name	Description	Address	Defined by
A descriptor with TCP/IP checksum extension is enabling TCP/IP checksum calculation. TCP/IP checksums 1 & 2 (16 bit) are calculated from **TCP Sum Start 1 & 2** up to the end of the packet. The checksums are written to **TCP Sum 1 & 2.** Note: An FCS appended to the frame by the XMACII is also added to the checksum. If the calculation is not ending 16-bit-aligned at the end of the packet, the missing byte is seen as 0. Registers holding these values are updated only, if **STF (Start of Frame)** is set (except **TCP Sum 1 & 2**)				
31:16	**TCP Sum 2**	Checksum 2 Valid, if **Own** is cleared by the BMU and **EOF** is set.	base + 0x18	network interface card, if EOF is set
15:0	**TCP Sum 1**	Checksum 1 Valid, if **Own** is cleared by the BMU and **EOF** is set.		
31:16	**TCP Sum Start 2**	Checksum 2, start position for calculation in bytes (16-bit aligned) counted from zero (first byte = 0)	base + 0x1c	host, if STF is set
15:0	**TCP Sum Start 1**	Checksum 1, start position for calculation in bytes (16-bit aligned) counted from zero (first byte = 0)		

Table 11: Receive Descriptor

STF, EOF

When receiving frames, i.e. transferring a frame from the buffer memory to the host memory, the STF, EOF bits are used to guarantee that the first fragment of a frame is written to a new contiguous host memory area. For new frames or fragments containing the start of a frame, the BMU accepts only receive descriptors with STF set to “1”. Receive descriptors with STF=“0” are relinquished to the host and the next receive descriptor is read. The design idea was to enable driver software to write frame headers in the memory areas where appropriate e.g. to the start of new contiguous buffers.

EN_IRQ_EOB

The design idea for the EN_IRQ_EOB interrupt was to give the driver software control over fragments from received frames e.g. the software provides only small buffers for the frame headers only. The BMU reads a receive descriptor with the STF set to “1” pointing to a small buffer area in

the host memory. The first fragment, the frame buffer, is transferred, and the BMU reads the next descriptor. However, this descriptor is still owned by the host. Thus no more frame fragments are transferred. Driver software meanwhile interprets the frame header. Then the driver software sets up the receive descriptors, starting with the one just read by the BMU. With knowledge about the frame header, driver software can provide space for the rest of the frame as appropriate. Then driver software relinquishes the descriptor(s) to the BMU and gives a Start <xxx> command to the involved BMU. This causes the BMU to transfer the rest of the frame.

DEV_0

The receive descriptor's DEV_0 bit is used to have the BMU reading this descriptor performing all the operations but without actually transferring data to the host memory. Setting the DEV_0 flag prevents that data is on the PCI bus, however the entire BMU mechanism is handled as usual, e.g. depending on the interrupt mask, an interrupt may occur.

Check

The check bits are provided to control the integrity of the descriptors and the BMU transfer mechanism. Depending on the setting TCP/IP hardware checksumming is enabled or disabled.

The following is important for driver software:

Take care to write the 32 bit containing the Own bit at last in a single action thus relinquishing ownership to the BMU. All other settings in a descriptor must be written before relinquishing ownership.

The hardware behavior:

On descriptor reads, RBCTRL, which is holding **Own**, is read first. If **EOF** is set, **RFSW** and **TCP Sum 1 & 2** are written before and a dummy read is inserted (if enabled by **En Dummy Read** in **Our Register 2**).

Note: Writing RBCTRL, data, which are defined by the host logically, are written back with their original values.

4.2.4 Transmit Descriptor

The transmit descriptors have the same structure for each of the four transmit queues:

Bit	Name	Description	Address	Defined by
TBCTRL Transmit Buffer Control				
31	**Own**	**Own** indicates that the descriptor is owned by the host (**Own** = 0) or the **network interface card** (**Own** = 1). The host sets **Own** for relinquishing the descriptor. The network interface card clears **Own** after reading the buffer pointed to by the descriptor entry. Once the network interface card or the host has relinquished the ownership of a descriptor, it must not change any field in the descriptor.	base	host network interface card
30	**STF** (Start Of Frame)	**Start of Frame** indicates that this is the first descriptor used by the **network interface card** for this frame. True, if = 1		host
29	**EOF** (End Of Frame)	**End of Frame** indicates that this is the last descriptor used by the **network interface card** for this frame. If **En IRQ EOF** and **EOF** are set, an interrupt **IRQ EOF** is generated after relinquishing this descriptor. True, if = 1		host
28	**En IRQ EOB** (Enable Interrupt on End Of Buffer)	If **En IRQ EOB** is set, an interrupt **IRQ EOB Rx** is initiated after relinquishing this descriptor. True, if = 1		host
27	**En IRQ EOF** (Enable Interrupt on End Of Frame)	enable **IRQ EOF**, see **EOF**. True, if = 1		host
26	**St&Fwd On** (Store and Forward On/OFF)	A frame is forwarded from RAMbuffer only, if the complete frame is in RAMbuffer. On, if = 1 Updated only, if **STF** is set.		host, if STF is set

Table 12: Transmit Descriptor

Bit	Name	Description	Address	Defined by
25	**Dis CRC** (Disable CRC)	Dis CRC is forwarded as TxCRCDisable to the XMAC: XMACII is not appending CRC to this packet. Updated only, if **STF** is set.		host, if STF is set
24	**SW<0>** (Software <0>)	May be used for software purposes. No effect on hardware		host
23:16	**CHECK/Opcode**	If the BMU is reading a descriptor owned by the BMU with an invalid **CHECK/Opcode**, an interrupt **IRQ CHECK TX** is generated and no further descriptors are read (supervisor state machine defaults to Idle). Opcodes: 0x55: Default descriptor 0x56: Descriptor with TCP/IP checksum extension		host
15:0	**TBBC** (Transmit Buffer Byte Count)	Transmit Buffer Byte Count. The host is defining the number of data to read from host memory. The BMU is writing back the number of effectively transferred data. There should be no difference except at EOF.		host network interface card
31:0	**NTDADR** (Next Transmit Descriptor Address)	Next Transmit Descriptor Address, lower 32-bit Transmit descriptors must be 8-byte aligned enabling 64-bit access by the BMUs	base + 0x4	host

Table 12: Transmit Descriptor

Bit	Name	Description	Address	Defined by
		(The upper 32-bit of the descriptor address are set once at initialization time for each queue in the control register file i.e. transmit descriptors are restricted to be in a joint 4 GByte address space, which means no severe restriction since the 4 GByte descriptor address space can be anywhere in the 64-bit address space. Transmit buffers however can be addressed anywhere in the 64-bit address space on a per descriptor base. Both the lower 32 bit and the upper 32-bit address is specified in the descriptor as shown below. I.e. data can be distributed over the entire 64-bit address space wherever the driver wants to have them.)		
31:0	**TBADR LO**	Transmit Buffer Address, lower 32-bit	base + 0x8	host
31:0	**TBADR HI**	Transmit Buffer Address, upper 32-bit	base + 0xc	host
31:0	**TFSW**	Transmit Frame Status Word as defined by the MAC. Appended to transmit data of this buffer, if **EOF** is set. This is a placeholder, because XMACII is not expecting a Transmit Frame Status Word. Default value should be 0. In loopback mode Transmit Frame Status Word has to be set to the expected Receive Frame Status Word from XMAC.	base + 0x10	host, if EOF is set

Table 12: Transmit Descriptor

Bit	Name	Description	Address	Defined by
TCP/IP checksum extension				
A descriptor with TCP/IP checksum extension is enabling TCP/IP checksum calculation and insertion. **St&Fwd On (Transmit Rambuffer Control Register**) overwrites **St&Fwd On** on a per packet base. I.e if **St&Fwd On** is set in the transmit RAMbuffer control register the setting in the descriptor is ignored. To switch **St&Fwd On** for each descriptor separately, **St&Fwd On** in the transmit RAMbuffer control register must be disabled. TCP/IP checksum(16 bit) is calculated from **TCP Sum Start** up to the end of the packet. **TCP Sum Offset** is added, the checksum is written to **TCP Sum Write.** If the calculation is not ending 16-bit-aligned at the end of the packet, the missing byte is seen as 0. Registers holding these values are updated only, if **STF** is set.				
31:16	**Reserved**	Value = 0	base + 0x14	host, if STF is set
15:0	**TCP Sum Offset**	Checksum, start value		
31:16	**TCP Sum Start**	Checksum, start position for calculation in bytes (16-bit-aligned) counted from zero (first byte = 0)	base + 0x18	host, if STF is set
15:0	**TCP Sum Write**	Checksum, write position for checksum in bytes (16-bit-aligned) counted from zero (first byte = 0)		
31:0	**Reserved**	Value = 0	base + 0x1c	host

Table 12: Transmit Descriptor

On descriptor reads, **TBCTRL**, which is holding **Own**, is read first. If the BMU is relinquishing the descriptor, **TBCTRL** is written as 32-bit word.

Note: Writing **TBCTRL**, data, which are defined by the host logically, are written back with their original values.

4.3 Data on the PCI Local Bus

Data transfer over the PCI bus is controlled by the BMUs which are build upon the Bus Interface Unit (BIU). The BIU comprises the target interface as described within section 3.0 *Programming Interface* and the bus master interface. The PCI transfer types implemented in the "**SK-NET GENESIS <x>**" are described below.

Though this manual is primarily for software development, the following kind of information may be usefull for diagnosis software, for troubleshouting using PCI bus analyzers and exercisers.

Burst Transfer

The default transfer bus master transfer type of the "**SK-NET GENESIS <x>**" is burst transfer, unless disabled by **Disable Burst**. The maximum burst size on the PCI bus is defined by the watermarks. Watermarks indicate the filling level of the PCI FIFOs. They can be configured for each FIFO independently in the control register file. The watermarks should be higher than two cache line sizes with a maximum of 1536 bytes (192 64-bit words).

64-Bit

If installed in a 64-bit slot and **UseData64** is set to 1, the state machine tries to perform full 64-bit accesses (Bit2:0 of the address forced to 0).
If the target is a 32-bit device, the upper 32 bits are transferred on the second data phase, and all following data phases use only the lower 32 bits of the bus. If a 32-bit target issues a target disconnect with data after the first data phase (which may not have contained valid bytes for a misaligned access), the state machine restarts on the upper 32-bit word address with a 32-bit access.
If installed in a 32-bit slot or **UseData64** is set to 0 the state machine performs 32-bit accesses (Bit1:0 of the address forced to 0).
Usage of the 64-bit extension can be disabled generally by setting UseData64 in the control register file (Our Register 2) to 0 at initialization time. The assertion of REQ64# by the network interface card is then disabled and bus transactions are performed as if mounted in a 32-bit slot.

The rising edge of **RST#** latches **REQ64#** for slot size recognition and status bit **Slot Size**. If **Slot Size** is 0 (network interface card installed in a 32-bit slot), all signals of the 64-bit extension of the PCI bus are driven to a stable logic level by the network interface card.

The network interface card defaults to 64-bit data width and 64-bit addressing. 64-bit addressing (Dual Address Cycle) is used only, if the upper 32 bit of the address isn't zero.
64-bit data width is disabled dynamically, when performing accesses to a 32-bit data width target and when performing put descriptor operations.

66 MHz Operation

The network interface card is able to perform all bus operations with 66MHz. Therefore the network interface card does not drive pin **M66EN** on the PCI bus. (As manufacturing option 66MHz operation can be disabled by connecting **M66EN** to GND. In this case the bit **66MHZCAP** in the Configuration **Status Register** must be reloaded with 0 from the Flash EPROM.)

Data Transfers

The transfer of a longer, misaligned frame may typically look like this:

The first transfer from any address is a burst with default read/write commands up to the next cache line size boundary. All following transfers are running between cache line size boundaries with optimized commands. The last transfer of the frame to any address is a burst with default read/write commands.

Normally each transfer is as long as the number of cache line sizes fitting into the watermark.

Access to the PCI bus is reallocated if a transfer is finished. Transfers are finished generally after moving the number of data given by the watermark or below in some exceptional cases (end/start of frame or buffer - latency counter expired).

The ownership of the bus is released with several reasons:

- ❏ Number of bytes to transfer is zero
- ❏ Cache line size boundary is reached and number of bytes to be transferred is below cache line size (exceptions: EOF,STF)
- ❏ Latency counter expired

The master sequencer state machine is controlled by giving address, guaranteed number of bytes to be transferred (plus minor additional informations) on a per cycle base from one of the queues.

Transfer Cycles

The number of transferred bytes is reported on a per cycle base. Supported commands are **Memory Write, Memory Write And Invalidate**, **Memory Read, Memory Read Line** and **Memory Read Multiple**.

Memory Write And Invalidate is used instead of **Memory Write**, if the guaranteed number of bytes to be transferred is higher than Cache Line Size. **Memory Read Line** is used instead of **Memory Read**, if the guaranteed number of bytes to be transferred is higher than one Cache Line Size. If the guaranteed number of bytes to be transferred is higher than two Cache Line Sizes at least, **Memory Read Multiple** is used instead of **Memory Read Line**.
The commands **Memory Write And Invalidate**, **Memory Read Line** and **Memory Read Multiple** may be disabled individually by setting the appropriate bits in **Our Register 1** (Configuration Register File).

If a cycle is terminated by **Target** or **Master Abort,** this is reported to **RTABORT** or **RMABORT** (**Status Register**), Interrupt **IRQ_MASTER** and **IRQ_STATUS** are set and the master sequencer state machine is locked. This has to be solved by resetting the state machine (**Reset_Master)**.

A target retry is serviced by retrying the terminated cycle.

Test

There is a specific mode of master accesses preventing transfers of unwanted frames to the system memory. This is signaled by **DEV_0** of the currently serviced descriptor to the BIU.

In this mode, no real access to the PCI bus is requested, but the interface to the master backend is simulating ‘normal’ bus accesses.

4.4 PCI Arbiter

The PCI Arbiter handles concurrent requests from the transmit and receive queues for PCI bus acccess. For data transfer the “**SK-NET GENESIS <x>**” operates as bus master i.e. data transfer over the PCI bus is always initiated by the network interface card for both the transmit and the receive queues. While it is obvious for receive queues that a frame is transferred over the PCI bus immediately after receipt, the NIC’s bus mas-

ter state machine -the BMU- must be triggered by the driver to look for transmit data/descriptors ready to send[1].
However after initialization of a transfer the sending/receiving of a frame is done without further software interaction. All actions on the NIC are descriptor controlled (exception: error handling)

The PCI arbiter grants access to the various queues following toggling priority rules:

1. Access is granted to Transmit / Receive queues alternately
2. Access is granted to Asynchronous / Synchronous alternately
3. Access is granted queues of MAC 1/ MAC 2 alternately

The PCI Arbiter is organized as an binary tree where for each decision the path is toggled.

4.5 Buffer Management Unit (BMU)

The BMU operates on the “descriptor+buffer” chains in the host system RAM. It comprises four interacting state machines:

- ❏ Supervisor state machine
- ❏ Descriptor read state machine
- ❏ Descriptor write state machine
- ❏ Transfer state machine

Each state machine may be forced to idle individually by setting **Reset xxx SM** (SM is used as an abbreviation for “State Machine”).

4.5.1 Supervisor State Machine

The supervisor state machine is issuing requests to the descriptor read and write state machines and to the transfer state machines. These state machines are reporting the execution of their transaction to the supervisor.

The supervisor is requesting a descriptor read, if the command **Start xxx** is given. This may be done either by driver/diagnosis software or by the **Start xxx** polling mechanism. If the **Descriptor Poll Timer** is enabled, a descriptor read is initiated periodically as defined by the **Descriptor Poll Timer Init Value.**

The supervisor state machine automatically requests the next descriptor after servicing the current descriptor is finished. With that the supervisor state machine provides a continuous data transfer with minimum CPU load on the host system as long as “descriptor+buffer” units are ready for transfer.

---

1. There’s also a poll mechanism that looks for frames ready to transmit. However, the design intention is that the driver prepares the transmit descriptor(s) then triggers the network interface card to do the transfer actually. (The polling mechanism is to avoid problems due to loss of the transfer trigger - this was observed in some environments with previous similar BMUs - adding the poll mechanism proved to be a reliable fix for this.)

If the descriptor read gives the ownership of the buffer to the network interface card, the request for transferring data is issued to the tranfer state machine. If the transfer from/to the buffer is closed, a descriptor write is requested to relinquish the descriptor to the host system by setting the descriptor's Own bit.

4.5.2 Descriptor Read State Machine

The descriptor read state machine initiates a bus master access to read the current descriptor. For this the descriptor read state machine is loading the 64-bit address counter with **Descriptor Base Address Register upper:Current Descriptor Address**, sets the number of bytes to be transferred to 32 and issues a request to the master backend state machine.
The reported number of transferred bytes is counting down the byte counter, transfer is done, if the byte counter reaches zero.

Note: the 32-bit word holding the **Own** bit is read first.

4.5.3 Descriptor Write State Machine

The descriptor write state machine is running up to three different types of bus master transfers over the PCI bus. In any case the last action of the descriptor write state machine is to relinquish the descriptor back to the host i.e. to driver/diagnosis software control by writing the 32-bit word that holds the **Own** bit.

Writing the 32-bit word holding the **Own** bit:

The descriptor write state machine is loading the 64-bit address counter with **Descriptor Base Address Register upper:Current Descriptor Address**, sets the number of bytes to be transferred to 4 and issues a request to the master backend.
The reported number of transferred bytes is counting down the byte counter, transfer is done, if the byte counter reaches zero.

On **EOF** write the **Receive Status Word** then the 32-bit word holding the **Own** bit with a second transfer:

Only if **EOF** in a receive descriptor is set, the **Receive Status Word** is written first before writing the 32-bit word holding the **Own** bit
The descriptor write state machine is loading the 64-bit address counter with **Descriptor Base Address Register upper:Current Descriptor Address + 16**, sets the number of bytes to be transferred to 16 and issues a request to the master backend.
The reported number of transferred bytes is counting down the byte counter, transfer of the **Receive Status Word** is done, if the byte counter reaches zero.

If additionally **En Dummy Read** in **Our Register 2** is set, after writing the **Receive Status Word** a dummy read to the same location is performed before writing the 32-bit word holding the **Own** bit. **En Dummy Read** is intended to avoid problems due to caching of PCI transfer data within caching PCI bridges. The additional read request should cause the bridge to actually perform the previous request.

Finally **Current Descriptor Address** is loaded with **Next Descriptor Address**. Control is given back to the supervisor state machine.

4.5.4 Transfer State Machine

The transfer state machine operates on the actual data, the frames or frame fragments. The transfer state machine is loading the 64-bit address counter with the **Current Buffer Address upper:Current Buffer Address lower**, setting the number of bytes to be transferred to **Buffer Byte Count** and issues a request to the master backend state machine.

Transfer is done, if, while counting the reported number of transferred bytes, the byte counter reaches zero or the FIFO is reporting **EOF** (in the case of the receive queues).

Testing the State Machines

Each state machine may be stepped by the software through its states.If set to testmode, state decisions and transitions are invoked by setting the command **SM Step**.

4.6 PCI Transmit/Receive FIFOs

The PCI FIFOs synchronize the dataflow between the PCI clock and the NIC's clock domain. To handle even burst transfers, the FIFOs are organized with a depth of 199 64-bit words and a width of 72 bits (64 data bits + 8 parity bits).

Each PCI FIFO interacts with its BMU (and the PCI arbiter) on the PCI buffer side and the RAMbuffer interface (and the RAMbuffer arbiter) on the other side.

Accesses at the PCI side may be executed at any byte boundary. Accesses at the other side, the RAMbuffer, are 64-bit aligned. An hardware multiplexer cares that chunks of data frames are still contiguous after passing the PCI FIFO area.

For each FIFO a separate watermark may be defined. All flags are synchronized to the Host Clock as well to the PCI Clock. They are providing bus requests and length information to the master backends.
Each FIFO may be cleared by its individual **Reset FIFO**.

Receive FIFO

The receive FIFO, i.e. each FIFO of the two receive queues, is providing flags:

- **Almost Full**, based on 64-bit words
- **EOF**, if tag bit in FIFO
- **Watermark**, if watermark (in bytes) is reached

The **Almost Full** and the **Watermark** flag can be configured in the control register file.

PCI Side

The receive FIFO is requesting transfers, if it is filled with data from the network so that watermark is reached or if **EOF** is set, i.e. the FIFO contains the end of a frame. In these cases the receive FIFO initiates data transfer over the PCI bus from the BMU.

The guaranteed number of available bytes is signaled to the BMU's master backend state machine. Once initiated, this number is decreased with each transferred byte. The initial value is the watermark in bytes or the real content of the FIFO, if **EOF** is set. If a transfer over the PCI bus is broken, the transfer may only be requested again, if the conditions above are given.

This limits the length of a transfer out of the receive queue to a maximum given by the watermark and forces the master frontend to lock on cache line size boundaries (if watermark greater than a cache line size).

The receive FIFO is tracking the start and the end of a frame. The messages **STF** and **EOF** are written to the bit fields in the descriptor, if the first or last byte is clocked out of the FIFO.

If the last byte of a frame is clocked out of the FIFO, the receive status word is written to the descriptor and the end of frame is reported to the BMU's master backend.

RAMbuffer side

The receive FIFO signals to the RAMbuffer's read state machine, that it is ready to receive data. It is ready, if there's space left in the FIFO (**Almost Full**) and no **EOF** is in the FIFO.

Almost Full is a cycle based signal and derived from the request pointer. The request pointer is incremented, if the RAMbuffer interface is acknowledging a request by the read state machine.

Data is written to the FIFO and the write pointer is incremented, if the RAMbuffer interface is acknowledging the transfer.

The RAMbuffer's read state machine signals an **EOF** after the last word is written to the FIFO. At **EOF** it is expected, that the receive status is valid. The receive FIFO is requesting no further transfers, as long as **EOF** is in the FIFO. The receive status is bypassing the FIFO.

The receive FIFO signals to the read state machine, that it took over the receive status (after **EOF** left the FIFO).

Transmit FIFO

The Transmit FIFO is providing flags:

- **Empty**, based on qwords
- **EOF**, if End of Frame in FIFO
- **Watermark**, if watermark (in bytes) is reached

PCI Side

A transmit FIFO is requesting transfers, if space is remaining. The threshold is set by the watermark. The guaranteed number of free bytes is signaled to the master backend. Once initiated, this number is decreased with each transferred byte.
The initial value is the watermark in bytes. If a transfer is broken, the transfer may only be requested again, if the conditions above are given.

This limits the length of a transfer to one of the transmit queues to a maximum given by the watermark and forces the master frontend to lock on cache line size boundaries (if watermark greater than a cache line size).

The transmit FIFOs are tracking the start and the end of a frame.

While **EOF** is in the FIFO, no further data is clocked into the FIFO. More generally: only one frame can be in the FIFO.

RAMbuffer side

The transmit FIFO signals to the RAMbuffer's write state machine, if data is available (**Empty**). **Empty** is a cycle based signal and derived from the read pointer. The read pointer is decremented, if the RAMbuffer interface is acknowledging a request from the write state machine.

The transmit FIFO signals an **EOF** after the last word is written to the FIFO. With **EOF** the transmit status is valid. The transmit status is bypassing the FIFO.
The transmit status is held until the write state machine acknowledges, that it took over the transmit status (after **EOF** left the FIFO).

A part of the transmit status is expected to be valid with the first data written to the FIFO: **Insert Checksum**, **Dis CRC**, **St&Fwd**.

4.7 RAMbuffer and RAM Interface

Throughout this manual the expression RAMbuffer is used for preallocated contiguous areas in an external[1] buffer memory. RAMbuffer also means the control logic building a FIFO for each queue in the external buffer memory. Thus a maximum of 6 RAMbuffers may be allocated at driver initialization. The RAMbuffers build an interim storage that helps to avoid congestion, preventing receive overflows, or that is used for TCP/IP checksum insertion.
The receive RAMbuffer serves as buffer for packet bursts, if the PCI bandwidth is too low. If the PCI bandwidth is too low permanently, flow control (according to IEEE 802.3x implemented in the XMACII) has to prevent receive overflows. The transmit RAMbuffer serves as a buffer for keeping at least one max packet, if the TCP/IP checksum has to be inserted or if the Transmit Queue is switched to Store & Forward (recommended for all but high performance PCI systems only such systems may outperform the NICs hardware).

RAMbuffers are operated as FIFOs, and when looking at the data flow they can be thought of as configurable-depth FIFOs inserted between the PCI FIFOs and the MAC FIFOs.
However, there is some control logic required for access to these FIFOs build in the external memory: the **RAM interface**. The RAM interface is running read/write cycles from/to the external memoy controlled by an instance arbitrating the pending requests. Dataflow is controlled by the filling level of the FIFO and the availability of data/space at the adjacent PCI FIFO and the adjacent MAC FIFO.

The RAM interface operates as arbiter for read / write requests from the PCI FIFO on one side and the MAC FIFOs (via Packet Arbiter + Transmit Arbiter) on the other side.

4.7.1 External Memory

The external memory consists of 2 or 4 synchronous SRAMs, either "flow through" or "pipelined" ones, with a data width of 32 bit plus 4 bit parity cascaded for a datawidth of 64-bit.

64k x 36 or 128k x 36 parts may be used (giving up to 2MB).

1. external means here "outside the ASIC"

The actual memory size can be read out from the **Eprom Register <0>.** One of the configurations requires an offset different from 0x0 when addressing the external memory.

(For future parts, addressing is extended by one bit thus doubling the amount of addressable memory. However, this option is not foreseen on the PCBs of the current versions. The internal address width is 19 bit, the highest address bit is converted to CE1/CE2 to the external memory.)

The SRAMs are clocked with host clock. The SRAMs are controlled by **CS*** and **WE*.**
The RAM interface is running read/write cycles from/to the external buffer memory controlled by an instance arbitrating the pending requests.
Requests are served cycle based, subsequent reads and subsequent writes are executed with one access on each cycle (even with random addresses). On changes from read to write, a turnaround cycle is inserted.

The number of delay cycles for read data depending on the type of the SRAMs is controlled by pin strapping through the pin **FT MODE***.

Access to the external memory is controlled and executed by the RAM Interface. The max burst length of memory accesses is limited by timeout counters for each participant individually. Per default the same burst length should be used.

4.7.2 RAMbuffer

RAMbuffers cover multiples of contiguous 8 kB areas. Thus a RAMbuffer may hold several Ethernet frames. Each frame in the RAMbuffer is preceeded by the first two 64-bit words of its **Internal Status** transporting the XMACII Statusword and a pointer to EOF in the FIFO.

Depending on the settings of **St&Fwd** in the RAMbuffer control registers, a RAMbuffer may be operated in Store and Forward mode or Flow Through mode. Store and forward mode means each frame is written completely into a RAMbuffer and only after EOF has been received, the frame is forwarded. Store and forward is needed e.g. when the TCP/IP checksum on a transmit queue should be computed. For this the entire frame must be read first -this happens while writing into the RAMbuffer- then the computed checksum is inserted into the frame.
If no TCP/IP checksum is computed, the RAMbuffer may be operated in “flow through” mode . Then the packet manager is bypassing the internal status.

The write/read state machines are tracking STF/EOF, reading/writing data from/to the external memory and reading/writing status from/to the external memory and from/to the adjacent PCI/MAC FIFOs.

Read/Write accesses from/to the external memory are requested from the RAM Interface.

Address Generation/Flags

The RAMbuffer’s memory location is defined by the **RAMbuffer Start** and **End Address.**

The **Buffer Write Pointer** is incremented, if data is written.
The **Buffer Read Pointer** is incremented, if data is read.

If they are reaching the **End Address**, they wrap to **Start Address**. Thus a frame may start near the **RAMbuffer End Address** and while writing the **Write Pointer** wraps around to the **RAMbuffer Start Address** so that EOF is near the **Start Address.**
The "**SK-NET GENESIS <x>**" hardware cares that the **Write Pointer** never overruns the **Read Pointer**. This makes sure that there's no data overwritten in the RAMbuffer.
The **Read/Write Pointer** and **Start/End Address** have to be set by software at initialization in Non Operational Mode (but with cleared **Reset**).

☞ **A RAMbuffer MUST NOT be switched to operational mode with inconsistent pointers/addresses.**

On each read/write of data **Level** is updated. **Level** gives the actual number of data in multiples of 8 bytes in this queue (including 16 bytes status word per packet). **Level** is the base for the flags **Empty** and **Almost Ful**l.

Receive Queues only:

If **Level** reaches the value of **Upper Threshold/Pause** configured in the control register file, the signal **XmtPausePkt** of the related XMACII is asserted (if **EnPause** is set). **XmtPausePkt** stays asserted, until **Level** is falling below **Lower Threshold/Pause**.
Thus the level of the RAMbuffer is used to reduce the receive data stream by the XMACII sending PAUSE frames to the connected Gigabit Ethernet device.

If **Level** reaches **Upper Threshold/High Priority**, highest priority for this receive queue is requested from the transmit arbiter (synchronizing on EOF of transmit packets). If granted, requests for memory accesses are placed with highest priority.

This request is staying as long as **Level** is above **Lower Threshold/High Priority**.

Write Statemachine

The write state machine is writing data to the FIFO, if there's space left and if the adjacent PCI/MAC FIFO signals to have data.

First incoming data after EOF or initialization are interpreted as STF. The write state machine initiates incrementing of the **Write Pointer** by two as a placeholder for the internal status. The packet manager keeps the start address of the packet.

Data is written to the FIFO until the adjacent PCI/MAC FIFO signals **EOF/Internal Status** available.

The internal status (end address of the packet) is written to the start address of the packet. If STF is already read by the read state machine, the internal status is bypassed through the packet manager. If the status arrived, the **Packet Counter** is incremented. EOF is acknowledged to the adjacent PCI/MAC FIFO.

The write state machine of a transmit queue additionally writes the TCP/IP checksum to the memory at EOF. The write position is calculated from the packet length given by the internal status and the start address of the packet. The checksum is packed into a complete 64-bit word and also given by the source in the internal status.

Read Statemachine

The read state machine is reading data from the FIFO, if there are data available and if the adjacent PCI/MAC FIFO signals to have space.

The first data after EOF or initialization is interpreted as STF. The read state machine initiates reading the status, if available. The status (end address of the packet) is forwarded to the packet manager.

If **St&Fwd** (**Rambuffer Control** or **Internal Status**/transmit queue) is set. reading at STF is delayed until status is available.

Data is read from the FIFO until the **Read Pointer** reaches the end address of the packet. Note: The end address is available from the status read at STF or bypassed by the packet manager.

If status is available and the read accesses of the last 64-bit word is executed, EOF/**Internal Status** is forwarded to the adjacent PCI/MAC FIFO and the packet counter is decremented.

Further reading is suspended until the adjacent PCI/MAC FIFO acknowledges EOF.

Packet Manager

A packet manager[1] is observing the dataflow to/from the FIFOs on packet level. It is observing the transfer of the internal status metadata from/to the adjacent PCI/MAC FIFOs.

The packet manager holds the **Packet Counter** and registers for storing the **Internal Status** temporarily. The **Packet Counter** is counting the packet ends available in the FIFO.

4.7.3 RAM Interface

The RAM interface handles access to the RAMbuffers for the requesting FIFOs, the PCI FIFOs and the MAC FIFOs.

Arbiter

Each requestor, i.e. each of the attached FIFOs, has a dedicated request and ID line, the acknowledgement is broadcasted to all requestors/participants by the RAM interface arbiter. A requestor/participant is identifying the acknowledgment by its ID, which is also broadcasted. For more specific identifying needs, the address of the executed access is also broadcasted.
After an arbitration each acknowledged access is queued-in. The execution including read data is broadcasted to all requestors/participants. A queued-in request is served as long it keeps staying. This is limited by the programmable **Timeout Counter**. Rearbitration takes place, if a request is withdrawn or counted out by the **Timeout Counter**.

---

1. There are several packet managers on the **"SK-NET GENESIS <x>"** NIC. There are separate packet managers for each RAMbuffer FIFO and each XMACII FIFO. Packet Managers for RAMbuffer FIFOs and for XMACII FIFOs are slightly different.

Requests are acknowledged in the same cycle as controlled by the arbitration rules. On changes from read to write there is a delay of 2/3 cycles, on changes from write to read there is a delay of 1 cycle.

Arbitration Rules:

The default arbitration is following a rotating priority scheme.

Requestors:

Receive queue 1 write state machine
Receive queue 1 read state machine
Synchronous transmit queue 1 write state machine
Synchronous transmit queue 1 read state machine
Asynchronous transmit queue 1 write state machine
Asynchronous transmit queue 1 read state machine

Receive queue 2 write state machine
Receive queue 2 read state machine
Synchronous transmit queue 2 write state machine
Synchronous transmit queue 2 read state machine
Asynchronous transmit queue 2 write state machine
Asynchronous transmit queue 2 read state machine

RAM Random Access

The priority scheme may be modified temporarily by the packet arbiter.

The RAM Random Access is foreseen for testability. Similar to the RAM-buffer configuration, diagnosis software may configure an memory area for RAM Random Access to check RAM, read out frames, internal status word 1 and 2.

Parity:

Write data is checked for parity, if parity is missing parity is generated. Read data is checked for parity.

4.8 Transmit Arbitration

Transmit arbitration, i.e. rate control, operates on the transmit queues only. If configured, transmit arbitration can be used to give the synchronous transmit queue precedence over the asynchronous transmit queue. Transmit arbitration is located before the MAC FIFOs and the associated packet arbiter.
Here at the MAC FIFOs the two queues, the synchronous and the asynchronous queue of each XMAC, flow into one transmit FIFO. The rules of the transmit arbiter determine precedence. There is a transmit arbiter for each transmit MAC FIFO.
The transmit arbiter is controlling the access to the transmit MAC FIFO on a per packet base.

The transmit arbiter controlls the following:

- ❏ No packet mismatch in the transmit MAC FIFO - synchronous and asynchronous transmit packets keep strictly separated though flowing into a joint data path now.
- ❏ Rate control: the synchronous transmit queue can be assigned precedence over the asynchronous transmit queue for a limited number of transfers.
- ❏ Prevents transmit underruns: if there are reads from the RAMbuffer the related write to the MAC FIFO has the highest priority.

The read state machines are requesting transfers from the transmit arbiter, if there are transmit data available in the RAMbuffer (or a complete packet, if switched to Store & Forward mode) and if there's space left in the MAC transmit FIFO.
The read state machines are providing additional information e.g. Start of Frame (STF) and Complete Packet Available.

The transmit arbiter is changing grants only, if there is no transfer running or with the End Of Frame (EOF) of a running transfer.

The read state machines are requesting RAMbuffer accesses only, if they got a grant from the transmit arbiter.

4.8.1 Rate Control

Rate control can give precedence to the synchronous transmit queue over the asynchronous queue for a configurable fraction of transfer cycles. This means that data in the synchronous queue has a guaranteed available transmit bandwidth on the network interface card. Data on the synchronous queue will be transferred continuously though the NICs transmit bandwitdh may be exhausted by an overfilled asynchnronous transmit queue. This may be used by running applications characterized by high burstiness on the network over to the asynchronous queue while applications that require a steady network data flow transmit over the synchronous transmit queue. Strictly speaking, rate control is directly effective on the network interface card only. The traffic shaping effect of rate control may spread until an attached switch. Then the network configuration and the attached devices determine how effective rate control influences end to end data throughput and traffic characteristics.

The rate control of the "**SK-NET GENESIS <x>**" is determined by the settings of three register values: **Rate Control Start/Stop, EN/DIS Alloc** (EN = Enable, DIS = Disable, Alloc means Allocation)**,** and **Force Sync ON/OFF.**

Force Sync ON/OFF	Rate Control	EN Alloc / DIS Alloc	
off	stop	dis	Only the asynchronous transmit queue is forwarding data.
off	stop	en	Data on the asynchronous queue has precedence. However if there are no asynchronous requests pending, the synchronous queue is served.
off	start	dis	Data on the synchronous queue has precedence depending on the value of the limit counter. If the limit counter reached Zero and no asynchronous request is pending, synchronous requests are served only when the limit counter is reinitialised.
off	start	en	Data on the synchronous queue has precedence depending on the value of the limit counter. If the limit counter reached Zero and no asynchronous request is pending, synchronous requests are served.
on	stop	dis	Data on the synchronous queue has precedence. However, if there are no synchronous requests pending, the asynchronous queue is served.
on	stop	en	
on	start	dis	
on	start	en	

Table 13: Rate Control

(The figure gives some examples for the settings highlighted in the table above.)

For configurable rate control the transmit arbiter is holding an interval timer and a limit counter giving a maximum number of synchronous transmit cycles per time interval. The ratio of the limit counter versus the interval timer determines precedence of the synchronous queue over the asynchronous queue. The interval timer and the limit are programmable in the control register file. Both are downcounters that are restarted by the interval timer reaching zero.

The interval timer provides a periodically repeating time interval. The configured bandwith ratio is assigned for each interval. The limit counter counts synchronous transfer cycles from the configured initialization value down to Zero. When reaching Zero the limit counter stops. The counter is restarted with its initialization value each time the interval timer wraps around from Zero to the timer's initialization value. The idea is that the limit counter reserves a fraction of the available transfer time per time interval

to priorize the synchronous data traffic.
As long as the limit counter does not reach Zero the synchronous queue is given precedence over the asynchronous queue.

Requests of the low priority read state machine are granted only, if there is no request from the high priority read-state machine and if it's allowed to allocate free bandwidth.This is allowed for the asynchronous read state machine per default, for the synchronous read state machine it must be enabled explicitly by **En Alloc** in the control register file.

However the most important rule at this arbitration point:
Running transfers are not interrupted. Each request has to wait for a running transfer to reach EOF.

The figure illustrating the rate control behavior gives some examples for the rules that determine rate control. It may also give an idea that the time interval and the limit counter should be large compared with the transmit frame length. If the limit counter or worse the time interval are comparable with the maximum transmit frame length it becomes more and more difficult to predict the exact rate control behavior.

Note: because transfer cycles are counted, the limitation works on absolute numbers (bytes per interval). This means, if the throughput on the MAC FIFO is lower than the theoretical maximum of 106.25 MB/s, the percentage of transfers of the synchronous RAMbuffer increases.

For contiguous frame flow the transmit arbiter interacts with the RAM interface. For example it causes the related RAM interface write state machine to be set to the highest priority (prevent transmit underrun). The **HP async** and **En Alloc** flags are also forwarded to the RAM Interface in order to control the write state machines.
The transmit arbiter is furthermore synchronizing requests of a receive RAMbuffer for higher priority at frame level (see **Receive Rambuffer Lower/Upper Threshold/High Priority**).

Interval Timer, Limit Counter

The **Interval Timer** is a programmable downcounter with a resolution of 18.825 ns (derived from host clock).

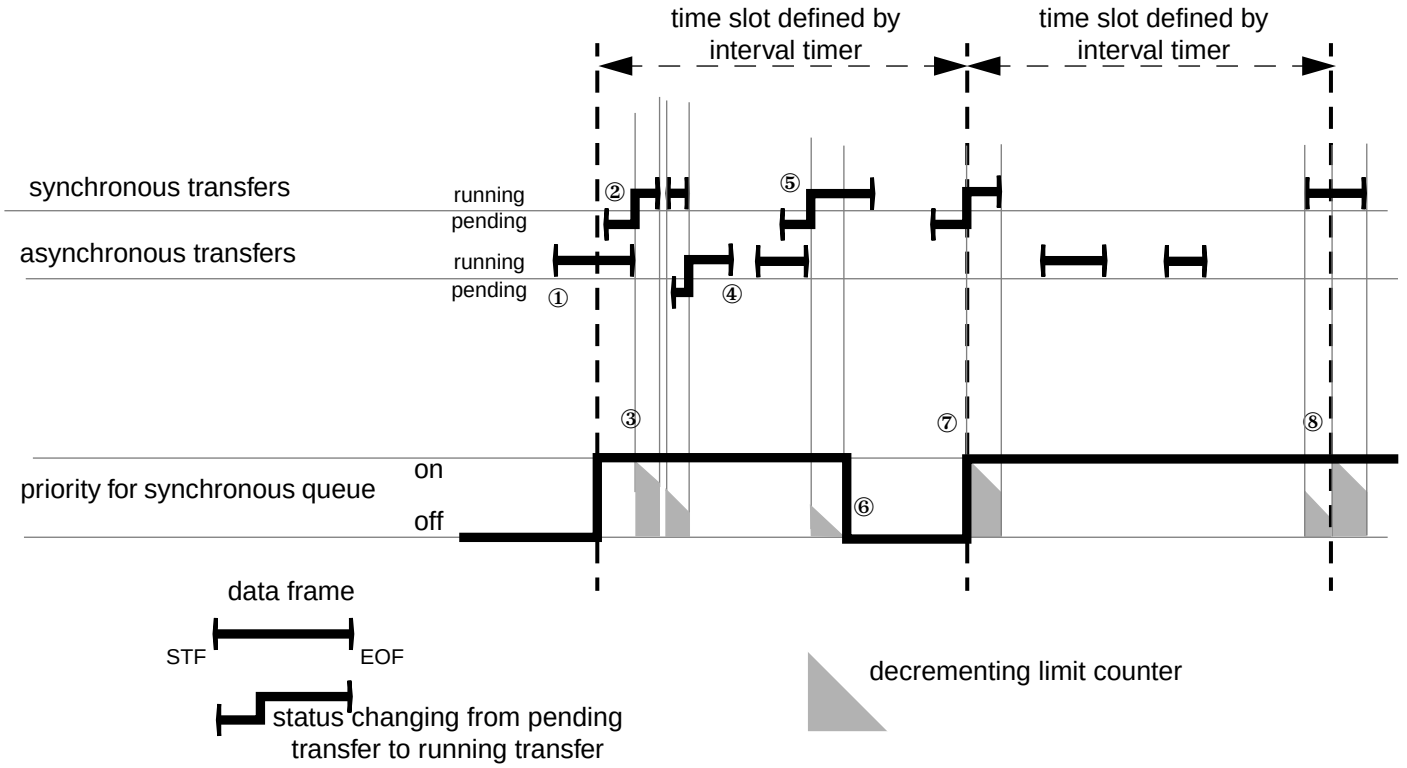
The Limit Counter is a programmable downcounter. It's counting the number of synchronous transfer cycles to the dedicated MAC FIFO.

The command **Rate Ctrl Start** loads **Interval Timer** and the **Limit Counter** with their **Init Value** and starts counting. Reaching ZERO the **Limit Counter** stops. If the **Interval Timer** is reaching ZERO or is loaded with ZERO the **Interval Timer** and the **Limit Counter** are reloaded with their **Init Value.**

The **Interval Timer** and the **Limit Counter** may be stopped by the command **Rate Ctrl STOP**.

Test

Test mode is switched on/off by the commands **xxx Test On/Off**. In test mode, clock pulses may be generated by software the commands **xxx Step**.



① An asynchronous frame is transferred completely - priority for synchronous queue is exhausted

② A synchronous frame request is pending - interval timer reached ZERO and was restarted with its initialization value - the limit counter was loaded with its initialization value too. -> priority for synchronous queues went to ON - nevertheless the pending synchronous transfer has to wait for termination of the running asynchronous transfer.

③ Asynchronous transfer terminated - the pending synchronous transfer is granted precedence and the limit counter is decremented - another short synchronous frame is transferred immediately while the priority for synchronous queues is on.

④ An asynchronous transfer request has to wait for the running synchronous frame transfer - since there's no pending synchronous request it can start immediately after the running synchronous transfer terminates.

⑤ A synchronous transfer is pending waiting for the running asynchronous transfer to terminate - then the synchronous transfer starts immediately - limit counter is decremented

⑥ The limit counter reaches ZERO --> priority goes to OFF until the interval timer wraps around - the running synchronous transfer keeps running until EOF

⑦ A synchronous request can start only when priority is on i.e. when the interval timer wraps around and the limit counter is reloaded with the initialization value. (With **EN Alloc** set, the synchronous transfer would have started immediatley since there was no concurrent asynchronous transfer.)

⑧ A synchronous transfer can start immediatley - priority is ON - when the interval timer wraps around the limit counter is reloaded - the running transfer decrements the reloaded limit counter.

4.9 Packet Arbitration

The packet arbiter is controlling the data flow between RAMbuffers and the MAC FIFOs on a per frame base. The packet arbiter is limiting the number of packets being transferred in parallel to 2.

Requests for receive packets have the highest priority. A request for a transmit packet is granted, if there is not running more than one transfer already and if there is no concurrent request for a receive packet.

There are timeout counters for each MAC FIFO (Receive queue MAC 1, transmit queue MAC 1, receive queue MAC 2, transmit queue MAC 2).

They are started, if enabled (**En Timeout Tx1 On**) and a transfer to/from the dedicated FIFO is started. If they are reaching **Zero**, they stop and generate an **IRQ Pkt Tout xxx**.
There are no further grants for that FIFO until **IRQ Pkt Tout xxx** is cleared (However granting max 2 packets to other FIFOs continues). **IRQ Pkt Tout xxx** is cleared by the command **Clr IRQ Pkt Tout xxx.** If **IRQ Pkt Tout xxx** is set, there may be more than 2 packets on the way shortly.
The timeout counters are counting Host Clock cycles (18.825 ns). Max timeout is 1.233696375 ms or 150.6 kB on a Gigabit Ethernet line.

Potential reasons for a timeout on receive are:

- ❏ A broken link
- ❏ receive RAMbuffer full (PCI/descriptors stuck)

Potential reasons for a timeout on transmit are:

- ❏ A broken link
- ❏ XMACII is transmitting Pause packets for flow control on receive

☞ **If the timeouts are not enabled, such a blocked transfer is also blocking the allocated bandwidth.**

Test

Test mode is switched on/off by the commands **Timeout Timer xxx Test On/Off**. In test mode, clock pulses may be generated by software the command **Timeout Timer xxx Step**.

4.10 MAC FIFOs

The MAC FIFOs are implemented in the ASIC by dual-ported RAMs operated as FIFOs. For each XMACII a dedicated pair of **Receive/Transmit MAC FIFOs** is implemented. The MAC FIFOs pipe the dataflow between the RAMbuffers and the XMACII II from XaQti. The XMACII II itself provides FIFOs for data exchange. The MAC FIFOs and the XMACII FIFOs are operated synchronously. The MAC FIFOs are used to translate data from 64-bit width at the RAMbuffer side in the ASIC to the 32-bit XMACII data interface.

Dataflow is controlled by the filling level of the MAC FIFO and the availability of data/space at the adjacent RAMbuffer or XMAC.

Each frame in the FIFO holds the first 2 64-bit words of the internal status transporting the XMACII statusword and a pointer to EOF in the FIFO. The FIFO's packet manager is bypassing the internal status, if a STF is read from a FIFO before EOF is written: status not available at read time.

The MAC FIFOs are running in flow through mode, which means, the first data may be forwarded before EOF is in the FIFO. The FIFOs are accepting more than one packet or fragments of more than one packet.

4.10.1 Receive

The **Receive MAC FIFO Registers** are giving access to nearly all resources. Most of them are accessible for testing/initialization purposes only. Only the **Clear IRQ No Status/Timestamp** are intended to be accessed on an operational **Receive MAC FIFO**.

Address Generation/Flags

The FIFO's depth is defined by the **FIFO End Address** (Start Address is **Zero** implicitly). The maximum end address is 0x3F.

The **Buffer Write Pointer** is incremented, if data is written.
The **Buffer Read Pointer** is incremented, if data is read.
If they are reaching the **End Address**, they wrap to **Zero**.
The **Read/Write Pointer** (and **Shadow** Pointers) and **End Address** have to be set by software at initialization in Non Operational Mode (but with cleared **Reset**).

☞ **A MAC FIFO MUST NOT be switched to operational mode with inconsistent pointers/addresses.**

On each read/write of data **Level** is updated. **Level** gives the actual number of data in multiples of 8 bytes in this queue (including 16 bytes status word per packet). **Level** is the base for the flags **Empty** and **Almost Full**.

Packet Manager

The packet manager is observing the dataflow to the FIFO on packet level and the transfer of the internal status to the receive RAMbuffer.

It's holding the **Packet Counter** and registers for storing the internal status temporarily. The **Packet Counter** is counting the packet ends available in the FIFO.

Write Statemachine

The write state machine is writing data to the FIFO, if there's space left and if the XMACII signals to have data.

The first incoming data after EOF or initialization is interpreted as STF. The write state machine initiates incrementing of the **Write Pointer** by two as a placeholder for the 2 x 64-bit internal status. The packet manager keeps the start address of the packet.
Data is written to the FIFO until the XMACII signals EOF/Receive Status available. Then the internal status (the end address of the packet) is written to the previously reserved start of the packet. This way the 2 x 64-bit of the internal status followed by the actual frame can be found in the MAC FIFO. (Remember it's not a real FIFO but an dual ported RAM operated as FIFO.)
However if the frame just flows through the MAC FIFO, i.e if STF is already read by the read state machine and forwarded to the corresponding RAM-

buffer, the internal status is bypassed through the packet manager. If the internal status arrived, the **Packet Counter** is incremented. EOF is signaled to the read state machine.

The Write SM is interfacing with the XMACII through XMACII signals **RcvValid**, **RxPktValid**, **RcvStatusValid** and **HostRcvRdy**. It is controlling a 64-bit input register converting incoming 32-bit data to 64-bit.

Frame metadata, the length of a frame, XMACII receive status and timestamp are stored in the internal status. Capturing the XMACII timestamp may be switched ON/OFF (**Time Stamp On/OFF**). However, this must be consistent with the XMAC's settings.

The XMACII is not delivering a receive status/timestamp on a receive overflow. Capturing the receive status/timestamp is timeout controlled by **Rx Status Timeout**/**Rx Timestamp Timeout**. On these timeouts undefined receive status/timestamp data is generated, **Rx Status Valid** / **Time Stamp Valid** flags in the internal status are not set. Interrupt(s) **IRQ No Status** / **IRQ No Timestamp** are generated.

A receive frame mast be treated as corrupted, if one of both **Status Valid** or **Time Stamp Valid** are not set.

☞ **Such a timeout event may generate some subsequent artificial receive packets with a length of 0 byte and Status Valid/Time Stamp Valid not set.**

Read Statemachine

The read state machine signals to the receive RAMbuffer, if there is data available. Data transfers are initiated by the receive RAMbuffer, if ready. The first data after EOF or initialization is interpreted as STF. The read state machine initiates reading the internal status, if available.

Data is read from the FIFO until the **Read Pointer** reaches the end address of the packet. Note: The end address is available either from the 2 x 64-bit internal status read at STF or it is bypassed by the packet manager.

If the internal status is available and the read accesses of the last 64-bit word is executed, EOF / internal status is forwarded to the receive RAMbuffer and the packet counter is decremented. Further reading is suspended until the receive RAMbuffer acknowledges EOF.

XMACII Patches

RxRdy Patch On/Off and **RxValid Timing Patch On/Off** are intended to adapt to Rev. B2 or Rev. C of the XMAC.For details see **Receive MAC FIFO Registers** in the control register file.

Pause Frames

The XMACII may be switched to automatic pause frame generation depending on the level of its receive FIFO.

Considering jumbo packets one can figure out scenarios where a receive overflow can not be prevented because the receive FIFO isn't deep enough. Pause frames may be additionally generated using **Upper/Lower Threshold/Pause Packets** (**Receive Rambuffer Registers**). If the receive RAMbuffer is filled up to **Upper Threshold**, signal XmtPausePkt of the related MAC is asserted (if **En Pause** is set). Signal XmtPausePkt

is deasserted, if the number of bytes is falling below **Lower Threshold**. The XMACII must be set to generating one Pause Packet only at each change of **Signal XmtPausePkt**.

This mechanism and the XMAC's automatic pause frame generation may be switched on in parallel.

Error Handling

The error handling is controlling the signal flow between the XMACII and the write state machine for error events.

Rx Status Timeout / Rx Timestamp Timeout and a receive overflow of the XMAC's receive FIFO is signaled by RxFIFOError. On a receive overflow the XMAC's receive FIFO is flushed by asserting FlushRxFIFO, if En Flush/Rx is set to "ON".

If there is a transfer of a frame in progress, this error event is treated like the end of a frame.

Rx Status Valid and Time Stamp Valid are not set in the internal status and interrupt(s) **IRQ No Status/IRQ No Timestamp** are generated. After completion FlushRxFIFO to XMACII is deasserted.

Parity Generation

For write data to the FIFO parity is generated and written to the FIFO.

4.10.2 Transmit

The **Transmit MAC FIFO Registers** are giving access to nearly all resources. Most of them are accessible for testing/initialization purposes only. None of them is intended to be accessed on an operational transmit MAC FIFO.

Address Generation/Flags

The FIFO's depth is defined by the **FIFO End Address** (Start Address is **Zero** implicitly). The maximum end address is 0x3F.

The **Buffer Write Pointer** is incremented, if data is written.
The **Buffer Read Pointer** is incremented, if data is read.
If they are reaching the **End Address**, they wrap to **Zero**.

The **Read/Write Pointer** (and **Shadow** Pointers) and **End Address** have to be set by software at initialization in Non Operational Mode (but with cleared **Reset**).

☞ **A MAC FIFO MUST NOT be switched to operational mode with inconsistent pointers/addresses.**

On each read/write of data **Level** is updated. **Level** gives the actual number of data in multiples of 8 bytes in this queue (including 16 bytes statusword per packet).
Level is the base for the flags **Empty** and **Almost Full**.

Packet Manager

The packet manager is observing the dataflow to the FIFO on packet level and the transfer of the internal status from the transmit RAMbuffer. It's holding the **Packet Counter** and registers for storing the internal status temporarily. The **Packet Counter** is counting the packet ends available in the FIFO.

Write Statemachine

The write state machine signals to the transmit RAMbuffer, if there's space left in the FIFO.

data is written to the FIFO if read from the transmit RAMbuffer. The first incoming data after EOF or initialization is interpreted as STF. The write state machime initiates incrementing of the **Write Pointer** by two as a placeholder for the 2 x 64-bit internal status. The packet manager keeps the start address of the frame.

Data is written to the FIFO until the transmit RAMbuffer signals EOF/ internal status available. The internal status (The end address of the frame) is written to the previously reserved start address of the frame. However if the frame is just flowing through the MAC FIFO, i.e. if STF is already read by the read state machine, the internal status is bypassed through the packet manager. After the internal status arrived, the **Packet Counter** is incremented. EOF is signalled to the read state machine.

Read Statemachine

The first data after EOF or initialization is interpreted as STF. The write state machine initiates reading the internal status or of it's place holder.

Data is read from the FIFO until the **Read Pointer** reaches the end address of the frame. Note: The end address is available either from the internal status read at STF or it is bypassed by the packet manager.

If the internal status is available and the read accesses of the last 64-bit word is executed, EOF is forwarded to the XMACII and the packet counter is decremented.

The read state machine is interfacing with the XMACII through the signals **XmtrRdy**, **TxValid**, **TxPktValid** and **TxCRCDisable**. It is controlling a 32-bit output register converting the internal 64-bit data to outgoing 32-bit data. **Dis CRC** (Transmit Descriptor / internal status) is forwarded to the XMACII as signal **TxCRCDisable** with the same timing than **TxPktValid**.

XMACII Patches

En Flush/Tx On/Off, **En Wait 4 Empty On/Off**, **Use AlmostFull On/Off**, **TxRdy Timing Patch On/Off**, **En Packet Recovery On/OFF** and **Wait after Flush** are intended to adapt to Rev. B2 or Rev. C of the XMAC.For details see **Transmit MAC FIFO Registers** in the control register file.

Error Handling

The error handling is controlling the signal flow between the XMACII and the read state machine on error events.

A transmit error at the XMAC's transmit side is signaled by **XmtPktError**. This causes **FlushTxFIFO to XMACII** becoming asserted. If there is a transfer of a frame in progress, transfer is completed normally by the XMACII Interface, but no data is transferred to the XMAC. After completion **FlushTxFIFO to XMACII** is deasserted.

Parity Checking

The parity of read data from the FIFO is checked. On an error, **IRQ Parity Error** is set. **IRQ Parity Error** may be cleared by the command **Clear IRQ Parity Error**.

4.11 Internal Loopback

Each pair of MAC transmit and receive FIFOs may be switched to loopback mode (**Transmit MAC Control** register in the contol register file). This can be used for measuring the PCI throughput or for testing the ASIC's internal data path.

In loopback mode transmit data from the MAC transmit FIFO are looped back to the MAC receive FIFO. No external control signals are generated, signals from the related XMACII are ignored.

The throughput at this point is 212.5 MBytes/s. This generates an maximum throughput of 425 MBytes/s on the PCI bus achieved by running concurrent master read and master write bursts.

Only one pair of MAC transmit and receive FIFOs may be switched to loopback mode. All devices, which are not needed for this loopback, must be inactive (resetted, non operational). All patch modes for any XMACII deviations must be switched off (**Transmit/Receive MAC FIFO Registers**).

Transmit descriptors should hold a transmit status. At the loopback point this is forwarded to the MAC receive FIFO as receive status, which means, it should reflect the expected receive status. There is no effect on any controlling logic related to the content of that status.
If forwarding timestamp is enabled (**Receive MAC Control**), the inverted receive status is forwarded as timestamp.

4.12 MAC Arbiter

The MAC arbiter is controlling the datapath between the MAC FIFOs in the ASIC and the XMACs receive and transmit FIFOs. This datapath is 32-bit wide. It is clocked with host clock, wich means a theoretical throughput of 212.5 MBytes/s can be achieved.

Two pairs of **MAC Receive/Transmit FIFOs** are requesting access to that datapath. Requests are arbitrated on a rotating priority scheme. After arbitration, a request is served as long as it keeps staying. This is limited by a **Timeout Counter** for each FIFO.
Rearbitration takes place, if a request is withdrawn or counted out by the **Timeout Counter**. Due to the timing requirements of the output drivers of the ASIC and of the XMACII rearbitration needs some time. Switching the datapath from a MAC transmit FIFO to a MAC receive FIFO takes 5 clock cycles without any data transfer.
Switching the datapath from a MAC receive FIFO to a MAC transmit FIFO takes 4 clock cycles without any data transfer.
If the **Timeout Init Values** are set to 63, the theoretical maximum throughput of 2 x 106.25 MB/s is reduced by 7% to 2 x 98.8 MB/s. The **Timeout Init Values** of 63 is adapted to the depth of the synchronizing FIFOs of the XMAC.
(If a future version of the XMACII with faster output enable timings is available, the switching time may be reduced to 4/3 potentially using **Fast OE On/Off.)**

All timeout counters of the MAC Arbiter are programmable. This gives the chance to tune/prioritize the burst length for the receive/transmit FIFOs. Per default the same burst length should be used.

For fixing XMACII Rev. B2 ringing problems there is a **Recovery Timer** for each FIFO. The **Recovery Timer** is started after a transfer of the dedicated FIFO. The FIFO doesn't get another frame until the **Recovery Timer** has been running down. The **Recovery Timers** may be switched on/off individually by the commands **En Rec xxx On/Off**.

4.13 XMACII

The host processor interface of the ASIC connects directly to the host TX/RX interface of the XMAC. This interface is running with host clock. For the XMACs a detailed documentation from XaQti Corporation exists, listing the main features, its operation and test features. The following lists the used pins of the XMACII:

Used Pins of the XMACII		
HOST TX Data (31:0)	TxByteEn(3:0)	TXpkt Valid
TxCRCDisable	TxFIFOAlmostEmpty	TxFIFOAlmostFull
FlushTxFIFO	XmtPausePkt	XmtPktError
XmtrRdy	TxValid	HostRxData(31:0)
RxByteEn(3:0)	RcvValid	RxPktValid
HostRcvRdy	RxFIFOAlmostEmpty	RxFIFOAlmostFull
RxFIFOError	RxOutDisable	FlushRxFIFO
NPData (15:0)	NPAddr (8:0)	NPR/W
NPCS	NPRDY	/NPINT
HOST_CLK	REF_CLK	NP_CLK
/RESET		
PODAT(9:0)	GTX_CLK	EN_COM_DET
PHY_LP_EN	PIDAT(9:0)	RBCLK,RBCLK_N
COM_DET	/LINK_SYNC[1]	GPInput[2]
V_{DD} (3.3V)	V_{CC} (5V)	V_{SS} (Ground)

Table 14: XMACII - Used Pins

1. /LINK_SYNC is connected to the GPINPUT signal - see XMACII Errata #2 March 1998 - the Errata sheet contains also detailed instructions for initialization and interrupt handling.

2. see previous footnote

XMACII Errata #2 gives instructions for hard- and software to avoid an inappropriate state which the XMACII may enter after receiving non-Idles over a period of time (not during normal packet reception) or when the XMACII is operated in manual mode (autonegotiation is disabled). The

"**SK-NET GENESIS <x>**" hardware is designed following this rules. The driver and diagnosis software has to follow the related instructions for initialization and interrupt handling.

Not used pins are not applicable in the design of the "**SK-NET GENESIS <x>**" NIC e.g. (NPData(31:16)) , they are for XMACII test purposes (TEST (2:0) or are not used for other reasons. Not used pins are usually bound to an approriate signal level and are available on the printed circuit board as hardware test points for diagnosis with hardware tools.

The node processor interface i.e. the interface to read/write the XMACs internal register is realized with 16-bit data width. Some of the XMACs special features that go beyond the IEEE specification are not used in the "**SK-NET GENESIS <x>**": e.g. FlushLastPkt, Transparent Mode are not supported.

4.13.1 Programming Interface

All 447 registers of each XMACII are mapped into the control register file. PCI bus accesses are translated to the XMACs using their node processor interface. The node processor interface is set to 16-bit mode. Each half of the 32-bit registers is mapped to one 32-bit word on PCI.

The interrupt line /NPINT of each XMACII is routed to the **Interrupt Source Register**. The reset line /RESET of each XMACII is controlled by **XMACII Reset.**

Data transfers from/to the MAC FIFOs of the ASIC to/from the transmit and receive FIFOs of the XMACs are running on the XMACs' host processor interface. All host processor interfaces of the XMACs are connected to one 32-bit bus. This bus is clocked with 53.12 MHz (HOST CLK) giving a maximum data rate of 212.5 MBytes/s.

There are some specific settings/options of the XMAC, which are mandatory, because the "**SK-NET GENESIS <x>**" is not supporting all of the XMACII options.

- ❏ GmiiMode (Hardware Configuration Register, bit 0) must be set to 0, because the "**SK-NET GENESIS <x>**" comes with a FC-0 interface for the PHY.
- ❏ SigStatChk (Hardware Configuration Register, bit 2) must be set to 0.
- ❏ EnableBigEndian (Mode Register, bit 2) must be set to 0. The ASIC's FIFO interface is not supporting big endian dataflow.

4.13.2 Receive Queue Performance

Receive data may be transferred through the MAC receive FIFO and through the receive RAMbuffer to the PCI receive FIFO, if XMAC's RxReqThreshold is reached or a complete frame has been received.

They are transferred to system memory, if the network interface card is the owner of the descriptor for the receive queue and the watermark in the PCI FIFO is reached.

The setting of RxReqThreshold is a trade-off between the acceptable delay on the receive path and the suppression of transfers of corrupted packets.

4.14 SERDES and Transceiver

Finally the the "**SK-NET GENESIS <x>**" is attached to the Gigabit Ethernet network via 802.3 z compatible transceivers. The 10-bit data stream from the XMACII is reduced to a serial 1-bit stream by the SERializer/DESserializer (SERDES).

5.0 More Resources

While the previous chapter was organized along the data path from the host system RAM to the Gigabit Ethernet network this chapter deals with the resources of the "**SK-NET GENESIS <x>**" that are not directly related to the data stream.

5.1 Flash EPROM

The 128KByte Flash EPROM may be mapped in the memory address space with sizes of 16 KByte, 32 KByte, 64 KByte or128 KByte. The page size is defined by the Page Size<2..0> bits hold by Our_Register1 in the configuration space.

Default is mapping of the full 128KByte Flash EPROM.

Mapping of the Flash EPROM is controlled by the EN_FPROM flag in Our_Register 1. If this flag is not set, the ROM Base Address Register is not presented to the configuration register file and the Flash EPROM may be not mapped into the memory address space.
Memory accesses to the Flash EPROM are read only. Write operations are completed normally on the bus and the data discarded.

If mapped, the base address is defined in the Expansion ROM Base Address Register. The Expansion ROM Base Address Register is also holding the page size and the ROMEN flag which controls enabling of the Expansion ROM. The mapped page is selected by setting PAGE<2..0> in Our_Register 1 .

The Flash EPROM loader provides another mechanism than memory mapping to read data, especially configuration data from the Flash EPROM.

The Flash EPROM may contain boot code or whatever but the last 16KByte sector is holding data for initialization of the NIC after power_on or **RST#**.

For programming of the Flash EPROM no additional 12V power supply and switching of the programming voltage is required. However

☞ **The Flash EPROM must be programmed carefully, according to the manufacturer's algorithms[1]. Any deviation may cause failure of or damage to the Flash EPROM.**

As Flash EPROM AMD's AM29F010 initially is used.

5.1.1 Flash EPROM Loader

The Flash EPROM loader is supporting:

- ❏ Loading of data after **RST#** from the Flash Eprom into the Configuration and Control Register File (where needed).
- ❏ Translation of the datawidth of 8-bit data width of the Flash EPROM to multiple byte/ 32-bit word memory read accesses from the bus.
- ❏ Programming interface using **Eprom Programming Registers**

1. If the AM29F010 Flash EPROM is mounted on the board, detailed information can be found in: "Flash Memory Products : 1992/1993 Data Book/Handbook" by Advanced Micro Devices (AMD).

The loader is capable of accessing potentially all registers in the **Control Register File** space. Register address and data is stored in 8-byte entries in the Flash EPROM.

The registers may be written with 32-bit, 16-bit, or 8-bit accesses.
The 8-byte entries are located on 8-byte boundaries from the end of the last 16k-sector of the Flash EPROM in decreasing order. Each entry is marked with a key.

If started, the loader is reading subsequent entries starting with the initial value of the **Eprom Address Counter** as long as a valid key is found. Loading is started by deassertion of **RST#** or the command **Load Eprom**. While loading, accesses to any resource of the network interface card are terminated by Target Retry Cycles.

The command **Load Eprom** is intended for testing purposes only. It is not recommended to reload the **Configuration Register File** using this command.

31:24	23:16		15:8	7:0	Address
key = 0x55	reserved	BE<3:0>	Address/upper	Address/lower	0x 1ffffc
Data<3>	Data<2>		Data<1>	Data<0>	0x1ffff8

Table 15: FRPOM Contents - Data Structure

Set to **Loader Test Mode**, the **Eprom Address Register** may decremented with the command **Loader Test Step**.

boot code:

32-bit read data is moved through byte registers and multiplexers from the 8-bit port of the Flash EPROM. Two, three or four byte accesses to the bus are serviced by subsequent byte accesses from the Flash EPROM storing the bytes in registers. There is no caching mechanism. This means, each read access invokes the related byte accesses from the Flash EPROM. The max. number of inserted wait cycles is 20 for an 32-bit word read access (CLK = 33MHz).

programming:

For programming purposes, defined read/write byte accesses to all Flash EPROM locations may be executed over the **Eprom Programming Registers** (control register file). These accesses are executable independent of the **Expansion Rom Base Address Register**, page size and page register.

Accessing the **Eprom Programming Data Register**, one byte is transferred from/to the Flash EPROM location defined by the **Eprom Programming Address Register.**

5.2 Interrupts

The **Interrupt Source Register** is holding the interrupts of all resources. Each interrupt is maskable by the **Interrupt Mask Registers**. All unmasked interrupts are or'ed and propagated to the bus interrupt line. An interrupt from a masked source can still be read from the **Interrupt Source Register**.

Interrupt generated by hardware checks are readable from the **Interrupt Hardware Error Source Register**. All unmasked interrupts are or'ed and propagated to the **Interrupt Source Register** a**s Interrupt Hardware Error.**

An interrupt is cleared and/or disabled as stated in the description of the related interrupt resource.

There's a **Special Interrupt Source Register** that is mirroring the **Interrupt Source Register** with special functionality adapted to typical software handling. Use of this register is recommended for driver software.

If the interrupt line **INTA#** is asserted, the read value is the same as in the **Interrupt Source Register**. If the interrupt line **INTA#** is NOT asserted, the read value is 0.

If the interrupt line **INTA#** is asserted, reading the **Special Interrupt Source Register** clears the **Interrupt Mask Register** (NOT the **Interrupt HW Error Mask Register**). As a result the interrupt line **INTA#** is deasserted.

5.2.1 Interrupt Moderation

The design idea behind interrupt moderation is to reduce interrupts by queueing non-critical interrupts. The interrupt queue is controlled by an timer that enables interrupts to be forwarded on the NIC's interrupt line periodically. All interrupts can be moderated by the **IRQ Moderation Timer**. Moderation is controllable individually for each interrupt by the **Interrupt Moderation Mask Registers.**

The **IRQ Moderation Timer** is a programmable 32-bit downcounter with a resolution of 18.825 ns (derived from host clock, Tmax = 80.85 s) for the usage as timebase for IRQ moderation.

The command **Interrupt Moderation Timer Start** loads **Interrupt Moderation Timer** with **Interrupt Moderation Timer Init Value** and starts counting.

Reaching ZERO or loaded with ZERO the **Interrupt Moderation Timer** is reloaded with **Interrupt Moderation Timer Init Value**.

The **Interrupt Moderation Timer** controls the assertion of the PCI bus line **INTA#** by gating the interrupts as defined by the **Interrupt Moderation Mask Registers**. If the **Interrupt Moderation Timer** is stopped or reaches ZERO, the gate opens and allows the masked interrupts to propagate to the bus. The assertion of **INTA#** caused by one of the masked interrupts is therefore delayed until the **Interrupt Moderation Timer** reaches ZERO. **INTA#** is kept asserted until the appropriate operation to clear the interrupt request is completed. The deassertion of **INTA#** is not affected.

The **Interrupt Moderation Timer** may be stopped by the command **Interrupt Moderation Timer STOP**.

While **HW Reset** or **SW Reset** asserted, the **Interrupt Moderation Timer** is stopped and the gate is closed. After releasing **SW Reset** the gate is initially open until the **Interrupt Moderation Timer** is loaded with a value other than ZERO and started.

Test

Test mode is switched on/off by the command **Interrupt Moderation Timer Test On/Off**.

In test mode, clock pulses may be generated by software via the command **Interrupt Moderation Timer Step**.

5.3 Parity Generation/Check

Parity is checked/generated on datapaths of receive/transmit data entering/leaving the ASIC. This includes the PCI Interface, the XMACII receive/transmit interface and the RAM interface.

On PCI parity checking/generating follows PCI specification for even parity on 32-bit words. All further parity checking/generating does even parity on bytes.

Internal byte based parity checking/generating

Byte based parity is generated on data entering the PCI transmit FIFOs and on data entering the MAC receive FIFOs. For internal status words written to the RAMbuffers, parity is generated by the RAM interface.

Byte based parity is checked on data leaving the PCI receive FIFOs and on data leaving the MAC transmit FIFOs. Parity is also checked on data read from the RAM.

Each parity checker is generating an interrupt. All parity interrupts are routed to the **Interrupt HW Error Source Register**. Note: Even if an **Interrupt Parity Error** is generated, running operations are continued.

Parity checking/generating on PCI as target

Read data parity is generated for read accesses to adapter resources in the ASIC.
Write data parity is checked for write accesses to adapter resources in the ASIC.

Address parity is checked for all address phases running on the bus.

If a write data parity error is detected, **Parity Error** is set. Bus signal **PERR#** is asserted, if **Parity Report Response Enable** is set.
If an address parity error is detected, **Parity Error** is set. Bus signal **SERR#** is asserted and **Signaled Error** is set, if **SERR# enable** and **Parity Report Response Enable** are set

Parity checking/generating on PCI as master

Write data parity is generated for all write accesses to system memory.
Read data parity is checked for all read accesses from system memory.
Address parity is generated for all address phases generated on the bus.

If a read data parity error is detected, **Parity Error** is set. Bus signal **PERR#** is asserted and **Data Parity Error detected** is set, if **Parity Report Response Enable** is set.

If on a write access **PERR#** is sampled asserted, **Parity Error** is set. **Data Parity Error detected** is set, if enabled by **Parity Report Response Enable**.

If **Data Parity Error detected** is set, interrupt **IRQ Master Error** is set.

If **Parity Error** is set, interrupt **IRQ Status** is set (see also **Interrupt Register**).

5.4 Reset Hierarchy

The power on reset **RST#** is setting **SW Reset**. **SW Reset** is setting **Master Reset,** all the device specific reset lines, statemachines, registers etc. (for details refer to the configuration register file) and the XMACII Resets (1 & 2).

Master Reset is resetting/releasing the queue arbiter and the switch logic and is synchronous input to the master sequencer state machine.

On a running network interface card **Master Reset** (and **SW Reset**) should be set only after a **Master Stop/Done**.

XMACII Resets (1 & 2) are propagated to the external XMACs.

☞ **If SW Reset is set, only the Control Register may be accessed. SW Reset must be cleared, before clearing all device specific Resets (even Master Reset in the Control Register).**

The Resets in the BMU Control/Status Registers are intended to be cleared/released in one step (they are split for testing purposes only).

Some devices are coming with an operational mode. They are set to operational mode OFF by their individual **Resets**.

The non operational mode is intended for initialization of all settings (pointers, switches etc.) of such a device. All these settings are not allowed to be modified in operational mode.

☞ **If switched to operational mode all settings must be consistent.**

5.5 Clock Distribution

All clocks are derived from the PCI clock and an oscillator (106.25 MHz) by the ASIC. Clock distribution is done by the ASIC (exception: SERDES clocks are driven by the related XMACII).

The GE (Gigabit Ethernet) clock of each XMACII is the individually buffered version of the incoming 106.25 MHz clock. All further XMACII clocks are derived from the incoming 106.25 MHz by divisions, each clock is distributed by an individual buffer/signal line.

The ASIC is running with two clock domains:

- PCI Clock for all PCI operations which must or may be synchronous to the PCI clock (including external I^2C interfaces, Flash EPROM and the External Register).
- Host clock (53.12 MHz) for all operations with a fixed time base and all operations interfacing to the external XMACs and the external memory.

The PCI Clock is provided on an output pad for any patching purposes and may be switched ON/OFF (**Dis PCI Clock** in **Our Register 1**).

The skew of the buffered PCI clock is defined by **Skew/PCI<3:0>** in **Our Register 1**

On the initial version **Dis PCI Clock** is switched OFF by overwriting from the EPROM Loader. All operations not running with PCI Clock are synchronous to host clock.

The skew of the clocks of the XMAC's FIFO interface is controlled by **Skew/Host<3:0>** in **Our Register 1**.

5.6 Timer

The **Timer** is a programmable 32-bit downcounter with a resolution of 18.825 ns (derived from Host Clock, Tmax = 80s) for the usage as a fixed timebase.

The command **Timer Start** loads **Timer** with **Timer Init Value** and starts counting. Reaching ZERO or loaded with ZERO the **Timer** generates an interrupt **IRQ Timer** and is reloaded with **Timer Init Value**.

IRQ Timer is cleared by the command **Timer Clear IRQ**. Command **Timer Clear IRQ** is overwriting a concurrent internal interrupt (guaranteeing IRQ edges).

Note: In order to prevent IRQ pulses, command **Timer Clear IRQ** should only be issued, if **IRQ Timer** is pending or if the **Timer** is stopped.

The **Timer** may be stopped by the command **Timer STOP**.

Test

Test mode is switched on/off by the command **Timer Test On/Off**. In test mode, clock pulses may be generated by software via the command **Timer Step**.

5.7 LINK_SYNC Counter

The **LINK_SYNC Counter** is a programmable 32-bit downcounter. It's counting the assertions of the signal /LINK_SYNC of the dedicated XMAC. It may be used to confirm a working connection with the neighbor port.

The command **LINK_SYNC Counter Start** loads the **LINK_SYNC Counter** with **LINK_SYNC Counter Init Value** and starts counting.

Reaching ZERO or loaded with ZERO the **LINK_SYNC Counter** generates an interrupt **IRQ LINK_SYNC Counter** and is reloaded with **LINK_SYNC Counter Init Value**. **IRQ LINK_SYNC Counter** is cleared by the command **LINK_SYNC Counter Clear IRQ**.

The command **LINK_SYNC Counter Clear IRQ** is overwriting a concurrent internal interrupt (guaranteeing IRQ edges).

Note: In order to prevent IRQ pulses, command **LINK_SYNC Counter Clear IRQ** should only be set, if **IRQ LINK_SYNC Counter** is pending or if the **LINK_SYNC Counter** is stopped.

The **LINK_SYNC Counter** may be stopped by the command **LINK_SYNC Counter STOP**.

Test

Test mode is switched on/off by the command **LINK_SYNC Counter Test On/Off**. In test mode, clock pulses may be generated by software the command **LINK_SYNC Counter Step**.

5.8 LEDs

The network interface card is providing 1 NIC LED and 3 port LEDs at each port. The intended use and the position is shown in section 2.1.1 *LEDs*

The usage is software / driver controlled. This section deals with programming the LEDs.

Besides switched on/off, some LEDs may be set to blinking. All blinking LEDs are using the same blink source.

Blink Source

The **Blink Source Counter** is a programmable 32-bit downcounter with a resolution of 18.825 ns (derived from host clock, Tmax = 80s) for the usage as fixed timebase for **Blink Source** intended to be used as gating signal for LEDs, which are switched to blinking.

The command **Blink Source Counter Start** loads **Blink Source Counter** with **Blink Source Counter Init Value** and starts counting. Reaching ZERO or loaded with ZERO the **Blink Source Counter** toggles **Blink Source** and is reloaded with **Blink Source Counter Init Value**.

The **Blink Source Counter** may be stopped by the command **Blink Source Counter STOP**.

Test

Test mode is switched on/off by the command **Blink Source Counter Test On/Off**.

In test mode, clock pulses may be generated by software via the command **Blink Source Counter Step**.

5.8.1 NIC LED

This LED is intended to indicate that a driver is loaded. It can be switchend on/off by setting **LED<0>*** (**LED Register**).

5.8.2 Link LED

The behavior of the **Link LED** is controllable by software through the **Link LED Control Register**.

It may be set to Off, On, blinking, forwarding /LINK_SYNC of the dedicated XMACII w/o blinking.

On/Off	Forward	Blink	LED is
0	x	x	off
1	0	0	on

Table 16: Link LED

On/Off	Forward	Blink	LED is
1	0	1	blinking
1	1	0	forwarding /LINK_SYNC
1	1	1	forwarding /LINK_SYNC while blinking

Table 16: Link LED

The blink frequency is controllable through the **Blink Source Counter**.

5.8.3 Receive LED

The **Rx LED** (receive LED) is indicating, that there are packets received. On any start of an incoming packet, **Rx LED Counter** is (re)loaded with **Rx LED Counter Init Value** and starts counting. Reaching ZERO, **Rx LED Counter** stops counting.
While **Rx LED Counter** is running, **Rx LED** is switched on. The Rx LED counter is a programmable 32-bit downcounter with a resolution of 18.825 ns (derived from host clock, Tmax = 80 s)

Triggering **Rx LED Counter** and switching **Rx LED** is enabled/disabled by **Rx LED Counter Start/Stop.**

Test

Test mode is switched on/off by the command **Rx LED Counter Test On/Off**. In test mode, clock pulses may be generated by software via the command **Rx LED Counter Step**.

5.8.4 Tx LED

The transmit LED (**Tx LED**) has the same behavior as described for **Rx LED** on transmitted packets using the **Tx LED** register/counter set.

5.9 I^2C Interface

The I^2C interface is controlled either by software through the **Interface Register** or by hardware through the **I2C Control Register** and **I2C Data Register**. If hardware controlled I^2C accesses are used, the **Interface Register** must be set to inactive values (Clock = 1, Direction = 0, Data = 0).

The hardware controlled interface can be parameterized in several ways. The size of the target device of the I^2C access (and implicit the number of address bytes/bits to be used) and its devsel byte must be written together with the address to the **I2C Control Register**. If the **I2C Burst** bit is set, the I2C interface runs 4 byte bursts in page mode, assuming pages of 8 bytes. Invalid or erroneous HW controlled I^2C accesses, that don't complete, can be stopped by writing a 1 to **I2C Stop**. On completion of a hardware controlled I^2C access an interrupt **IRQ I2C Ready** is asserted.

5.10 Temperature/Voltage Sensor

As a manufacturing option a temperature/voltage sensor may be assembled. Its internal registers may be accessed through the I^2C interface in the **Interface Register** or the **I2C Control Register** and **I2C Data Register**.

The I^2C address of the sensor is 0b0101_000. As sensor National's LM80 is used initially.
The sensor's IRQ line (INT#) is connected to the ASIC's input **IRQ EREG***.

- ❏ Port IN<0> is measuring VCC (5V)
- ❏ Port IN<1> is measuring VIO (PCI
- ❏ Port IN<2> is measuring VDD (3.3 V -> ASIC, XMACs)
- ❏ Port IN<3> is measuring PLC_3V3 (SERDES)

Voltage:

The voltage range is 0 - 5,61V (8bit, resolution = 22mV) The tolerance is 1 LSB +/- 1,1%.

Temperature:

The sensor is placed near the PC bracket, such not giving the highest PCB temperature. A correction parameter has to be defined after some measurement.

6.0 Running the NIC

This chapter summarizes the sequence of events that are necessary to operate the **"SK-NET GENESIS <x>"** network interface card**.**

6.1 Initialization

Correct initialization is crucial for operation of the network interface card. The device is in reset state after power on. The following Reset sequence is mandatory.

- ❏ Release **SW Reset**
- ❏ Release **Master Reset**
- ❏ Release **XMACII 1 / XMACII 2 Reset**

Now configure all devices, the following sequence may be varied. However if a device has an individual Reset, this Reset must be released at first. After releasing the Reset, all registers are set to their required values.

If a device has an operational mode, it may be set to operational only after setting all registers.

☞ **Register settings are not allowed to be modified in operational mode.**

☞ **Any commands (e.g. Start xxx) are allowed only, if a device is configured completely.**

☞ **Refer to the XMACII datasheet and Errata for proper initialization of the XMACII.**

- ❏ Configure Timer
- ❏ Configure Blink Source Counter
- ❏ Configure Link_Sync Counter
- ❏ Configure Rx LED Counter
- ❏ Configure Tx LED Counter
- ❏ Configure Descriptor Poll Timer
- ❏ Configure I^2C Registers
- ❏ Configure MAC Arbiter

  There are settings, which must be consistent to settings of the XMACII (see receive/transmit MAC FIFO registers)

- ❏ Configure 4 MAC-FIFOs

  There are settings, which must be consistent to settings of the XMACII (see Receive/Transmit MAC FIFO registers)

- ❏ Configure Packet Arbiter
- ❏ Configure 2 Transmit Arbiters
- ❏ Configure 6 RAMbuffer
- ❏ Configure RAM Interface
- ❏ Configure 6 BMUs
- ❏ Initialize descriptor mechanism (BMU)
  Setup descriptors in system memory (at least for queues, which are intended to be used, one descriptor minimum).
  Release the resets of all components of the queues, which are intended to be used.
  Set **Current Descriptor Address Upper/Lower** to first descriptor in system memory.
- ❏ Configure interrupt handling
- ❏ Configure (2) XMACs
- ❏ Start the timers and counters
- ❏ Start the BMUs
- ❏ Enable the XMACII's transmit and receive state machines (MMU reg) when the link is synchronized

6.2 Data transfers

6.2.1 Receive

One way to manage receive frames may look as follows:

- ❏ The network interface card is not owner of the current descriptor.
- ❏ Software is releasing the current descriptor to the network interface card and issuing **Start xxx** to the receive queue
- ❏ The subsequent descriptors are not relinquished to the network interface card.
- ❏ **Start xxx** is initiating reading the current descriptor with the **Own** bit set to the network interface card.

- ❏ The first data received are moved to this buffer.
- ❏ If the buffer is filled or **EOF** is reached, interrupt **EOB** is set after relinquishing the descriptor.
- ❏ After controlling the header of the received frame, software defines the destination for further data of this frame by defining and releasing the subsequent descriptor(s).
- ❏ data is transferred, until **EOF** is reached or no further descriptor is released to the network interface card.

Transfer of an unwanted frame may be prevented by setting **Dev 0**. The network interface card is handling this like 'normal' transfers, except that the data is not transferred really to system memory in order to prevent waste of PCI bandwidth.

6.2.2 Transmit

Transmit data may be transferred from system memory, if space is remaining in the PCI transmit FIFO. Transmit data is forwarded through the transmit RAMbuffer and the MAC transmit FIFO to the XMACII's transmit FIFO. The XMACII is initiating transmitting, if data in the XMACIIs transmit FIFO reach TxReqTreshold or a complete packet is in its transmit FIFO.

The typical way to manage transmit frames may look as follows:

- ❏ The network interface card is not owner of the current descriptor.
- ❏ Software is releasing the transmit descriptor(s) to the network interface card and issuing **Start xxx** to the transmit queue.
- ❏ **Start xxx** is initiating reading the current descriptor with the **Own** bit set to the network interface card.
- ❏ While space is remaining in the transmit queue, data is moved from system memory on a per descriptor base.
- ❏ If a buffer is emptied or **EOF** is reached, interrupt **EOB** is set after relinquishing the descriptor.
- ❏ Data is transferred, until **EOF** is reached or no further descriptor is released to the network interface card.

6.2.3 Stop a Receive Queue

To stop a receive queue the following sequence is proposed:

- ❏ Stop XMACII's receive state machine (MMU Register)
- ❏ Wait for MAC FIFO empty (**Level**)
- ❏ Wait for Receive RAMbuffer got empty (**Level**)
- ❏ Wait for PCI FIFO empty (**Level**)
- ❏ Stop releasing descriptors or get back unused descriptors

6.2.4 Stop a Transmit Queue

Note: There's a risk, that the last packet comes with Status invalid

To stop a transmit queue the following sequence is proposed:

- ❏ Stop releasing descriptors or get back unused descriptors

- ❏ Wait for PCI FIFO empty (**Level**)
- ❏ Wait for transmit RAMbuffer empty (**Level**)
- ❏ Wait for MAC FIFO empty (**Level**)
- ❏ Stop XMACII's transmit state machine (MMU Register)

7.0 Tests

The **"SK-NET GENESIS <x>"** comes with many registers and flags that are not needed for actual operation of the NIC but are made accessable for test and control purposes. Besides the signals and registers required internally that are made available for test / diagnosis software, some flags and registers are provided for testability. E.g. the Random RAM access that can be used by the software to verify contents of the RAMbuffers for tracing data frames and frame metadata. This chapter lists some of the test features of the **"SK-NET GENESIS <x>"** that were not mentioned yet e.g. the internal loopback path between the MAC FIFOS in the ASIC, etc.

7.1 Testing BMU State Machines

Each of the 6 queues at the PCI side comes with three test registers. The test registers may be used to

- ❏ Stepping through the states of BMU state machines
- ❏ Decrement the byte counter
- ❏ Verify operation of the hardware multiplexer

7.2 State Machine - State Sequence

The register descriptions in the control register file give some information how to step to a BMU state machine. (Set test mode ON). The following lists the sequence of states that a state machine reaches when steping through:

Table 17: States of Receive State Machine

State	Q<3:0>
Supervisor SM	
Idle	0b0000
Res_Start	0b1000
Get_Desc	0b1011
Check	0b1001
Wait_Data	0b1101
Set_Pkt_Xfer	0b1111
Move_Data	0b0111
Put_Desc	0b0011
Res_Stop	0b0001
Set_Irq	0b1100

Table 17: States of Receive State Machine

State	Q<3:0>
Read Descriptor	
Idle	0b0000
Load	0b0001
Wait_TC	0b0011
Reset_EOF	0b0110
Wait_Done	0b0010
Transfer Data	
Idle	0b0000
Load	0b0001
Wait_TC	0b0010
Wait_Done	0b0100
Write Descriptor	
Idle	0b0000
Act_Buf_Length	0b0001
Wait_EOF	0b0011
Load_Stat_A	0b0010
Wait_Stat_TC	0b0110
Load_Dumr_A	0b1110
Wait_Dumr_TC	0b1111.
Load_Desc_A	0b0111
Wait_Desc_TC	0b0101
Load_Ndsc_A	0b0100
Wait_Done	0b1100

Table 18: States of Transmit State Machines

State	Q<3:0>
Supervisor SM	
Idle	0b0000
Res_Start	0b1000
Get_Desc	0b1011
Check	0b1001
Wait_Data	0b1101
Set_Pkt_Xfer	0b1111
Move_Data	0b0111
Put_Desc	0b0011
Res_Stop	0b0001
Set_Irq	0b1100
Read Descriptor	
Idle	0b0000
Load	0b0001
Wait_TC	0b0011
Wait_Done	0b0100
Transfer Data	
Idle	0b0000
Load	0b0011
Wait_TC	0b0010
Wait_Done	0b0100
Write Descriptor	
Idle	0b0000
Act_Buf_Length	0b0001
Res_OWN	0b0010

Table 18: States of Transmit State Machines

State	Q<3:0>
Load_Desc_A	0b0110
Wait_Desc_TC	0b0111
Load_Ndsc_A	0b1000
Wait_Done	0b1001

7.3 Hardware Multiplexer

For each queue there's a hardware multiplexer controlling the data flow into the PCI FIFOs. The multiplexer position is calculated on a per descriptor base.

A modulo-8 calculation is sufficient for determining one out of eight positions.
If a descriptor with the **Own** bit set to the network interface card is read, the new position is:

Mux<2:0> = Buffer start address - Virtual RAMbuffer address

The virtual RAMbuffer address (**VRam**) is updated to the current length of the transferred frame, if the descriptor is released:

VRam<2:0>:= VRam<2:0> + Buffer length

VRam is initialized to zero on reset or after transferring a buffer holding **EOF**.

For testing/controlling purposes, **VRam<2:0>** may be written, **Mux<2:0>** is only readable. If **VRam<2:0>** is modified, **Mux<2:0>** should follow depending on the current buffer start address.

8.0 External Interfaces

This chapter gives some information about the NICs external interfaces:

- ❏ The PCI bus
- ❏ The Gigabit Ethernet transceivers

8.1 PCI

The system interface of the network interface card is fully compliant with the PCI Specification Rev. 2.1. It supports 32-bit or 64-bit bus master and 32-bit target transfers on the PCI with the 33 MHz or the 66 MHz bus timing modes.

PCI Signals

The following table shows the used PCI signals:

Group	Signal	I/O	Function
Address & Data 32 Bit Part	**AD[31:00]**	t/s	Multiplexed data and address lines
	C/BE[3:0]#	t/s	Bus command and byte enable lines
	PAR	t/s	Even parity over AD[31:0] and C/BE[3:0]
Address & Data 64 Bit Extension	**AD[63:32]**	t/s	Multiplexed data and address lines
	C/BE[7:4]#	t/s	Bus command and byte enable lines
	PAR64	t/s	Even parity over AD[63:32] and C/BE[7:4]

Group	Signal	I/O	Function
Interface Control	**FRAME#**	s/t/s	Cycle frame
	TRDY#	s/t/s	Target ready
	IRDY#	s/t/s	Initiator ready
	STOP#	s/t/s	Target STOP request, used by the network interface card only for target disconnect with/without data.
	DEVSEL#	s/t/s	Device select, asserted by the network interface card with medium **DEVSEL#** timing
	IDSEL	in	Initialization device select
	LOCK#	s/t/s	not used by the network interface card
	REQ64#	s/t/s	Request 64-bit transfer signal driven by the current bus master.
	ACK64#	s/t/s	Acknowledge 64-bit transfer signal driven by the current bus target.
Error Reporting	**PERR#**	s/t/s	Parity error, is asserted by the network interface card for all data parity errors detected, if enabled
	SERR#	o/d	System error signal is asserted by the network interface card for all address parity errors detected, if enabled
Arbitration	**REQ#**	t/s	Bus request, asserted by the network interface card to gain bus ownership, kept asserted until second last data phase of a transaction
	GNT#	t/s	Bus grant
System	**CLK**	in	PCI bus clock. The network interface card is working at any frequency from 0 - 66 MHz. The network is operable at any frequency from 25 - 66 MHz.
	RST#	in	Reset signal, brings the network interface card into a consistent state. All local resources on the network interface card are reset.

Group	Signal	I/O	Function
Interrupt	**INTA#**	o/d	The interrupt signal indicates an interrupt request from the network interface card to the system. The assertion and deassertion of **INTA#** is asynchronous to **CLK**. A pending request is cleared by the interrupt service of the device driver.
Additional Signals	**PRSNT[1:2]#**		The present signals indicate to the motherboard whenever the network interface card board is present and if it is present, the total power requirements of the network interface card. The default setting of **PRSNT[1:2]#** is 0b00 (7.5 W max).
JTAG	**TDI**	in	Test data In for JTAG boundary scan test path. The TDI pin is routed to the TDO pin on the network interface card. No further scan test functions are supported by the network interface card.
	TDO	out	Test data OUT for JTAG boundary scan test path. On the network interface card the TDO pin is routed from the TDI pin.

Used I/O pin description:

in	Input
out	Output
t/s	Tri-State
s/t/s	Sustained Tri-State (active low Tri-State with external pull-up resistor on the motherboard)
o/d	Open Drain

Target Transactions

If the network interface card is the target of a PCI bus operation, **DEVSEL#** is asserted by the network interface card with medium timing.

If the network interface card is the target of a PCI bus burst operation, the operation is terminated with **disconnect with data** (**TRDY#** and **STOP#** asserted) in the first data phase.

If the network interface card is the target of a PCI bus operation in the reset or initialization phase of the network interface card, it responds with a **target retry** (**STOP#** asserted, **TRDY#** deasserted).

On read operations all data lines **AD[31:0]** are driven independent from the **C/BE[3:0]#** lines.

Write operations to reserved registers in the configuration space or control register file are treated normally on the bus with data discarded. A read operation to a reserved register is serviced normal with data value of 0 returned.

Slave Configuration Read/Write

The PCI configuration space of the network interface card is readable and writable by an external PCI bus master via configuration read and write accesses.

A configuration access to the network interface card is indicated by the appropriate command on the **C/BE[3:0]#** lines (0b1010 for read, 0b1011 for write), **IDSEL** asserted and **A[1:0]** set to 0b00 (Type 0 configuration cycle) during the address phase. In the data phase the byte enable signals on the **C/BE[3:0]#** lines indicate, which bytes of the transfer are valid.

Accesses to the configuration space are allowed with 8-bit, 16-bit and 32-bit transfers.

The register layout and the functions of the PCI configuration space registers is described in chapter "**Configuration Register File**".

Slave I/O Read/Write

The control register file of the network interface card can be mapped into the 32-bit I/O space enabled by **En IO Mapping** in the **Our Register**. Than, the base address of the control register file is determined by the contents of the **Base Address Register (2nd)** in the configuration space.

An I/O access to the network interface card is indicated by the appropriate command on the **C/BE[3:0]#** lines (0b0010 for read, 0b0011 for write) and the I/O Address on **AD[31:0]** lines during the address phase. In the data phase the byte enable signals on the **C/BE[3:0]#** lines indicate, which bytes of the transfer are valid. The network interface card doesn't check for inconsistent **AD[31:0]** / **C/BE[3:0]#** combinations.

The Control Registers have to be accessed with the appropriate data width (8-bit, 16-bit or 32-bit) as the register is defined.

Slave Memory Read/Write

The memory mapped I/O resources of the network interface card and, if selected with **En Eprom** in **Our Register 1**and **ROMEN** in the **Expansion Rom Base Address Register**, the Expansion ROM (Flash EPROM) can be accessed via memory read or write cycles. The memory base address for the memory mapped I/O resources is determined by the contents of the **Base Address Register (1st).** The memory base address of the Expansion ROM is determined by the contents of the **Expansion Rom Base Address Register**.

A memory access to the network interface card is indicated by the appropriate command on the **C/BE[3:0]#** lines (0b0110, 0b1100 or 0b1110 for read, 0b0111 or 0b1111 for write) and the memory address on **AD[31:0]** lines during the address phase. In the data phase the byte enable signals on the **C/BE[3:0]#** lines indicate, which bytes of the transfer are valid.

Memory Read Line/Multiple are aliased to Memory Read, Memory Write and Invalidate is aliased to Memory Write by the target section of the network interface card.

The control registers have to be accessed with the appropriate data width (8-bit, 16-bit or 32-bit) as the register is defined.

Bus Acquisition

With **REQ#** asserted the network interface card requests the PCI bus from the system arbiter and waits for **GNT#** asserted on the bus. When it detects **GNT#** asserted and the PCI bus in idle state it starts with master bus cycles under control of the Buffer Management Unit (BMU).

REQ# is kept asserted until the begin of the second last data phase for burst operations.

REQ# is kept asserted until the begin of the data phase for single cycles.

Bus Master DMA Transfers

The network interface card uses the full set of PCI memory transaction commands for Bus Master DMA Transfers. It always uses the appropriate command according to the amount of data to be transferred. That is:

Memory Read/Write (Single/Burst) for misaligned data up to the next cache line boundary and incomplete cache lines.

Memory Read

Memory Read Multiple

Memory Write and Invalidate for 1 and more complete cache lines.

Target Initiated Termination

If the network interface card while transferring data as bus master receives a **target retry**, it deasserts **REQ#** for at least 2 **PCICLK** periods and repeats the transaction immediately afterwards.

If the network interface card while transferring data as bus master receives a **target abort** (**STOP#** asserted, **DEVSEL#** deasserted), it terminates this cycle with **FRAME#** deasserted and **IRDY#** deasserted one clock later and sets the **RTABORT** bit in the **Config Status Register**. If not masked, an interrupt is generated on the PCI bus with the IRQ status bits **IRQ Master Error** and **IRQ Status** set.

After a master or target abort condition, the busmaster state machine is forced in idle state until the **RMABORT** and the **RTABORT** bits in the **Config Status Register** are cleared.

Master Initiated Termination

If the network interface card tries to access to a PCI target and there is no response (**DEVSEL#** remains deasserted) after the Devsel Timeout has expired the master must assume that the slave does not respond. The master terminates this cycle with **FRAME#** deasserted and **IRDY#** deas-

serted one clock later and sets the **RMABORT** bit in the **Config Status Register**. If not masked, an interrupt is generated on the PCI bus with the IRQ status bits **IRQ Master** and **IRQ Status** set.

After a master or target abort condition, the busmaster state machine is forced in idle state until the **RMABORT** and the **RTABORT** bits in the **Config Status Register** are cleared.

A normal Master transaction is terminated by the network interface card when the intended data has been transferred or when the Latency Timer has expired and the **GNT#** is deasserted.

The commands Memory Write and Invalidate, Memory Read Line and Memory Read Multiple ignore the Latency Timer until a cacheline boundary is reached.

8.2 Gigabit Ethernet Interfaces

The initial realization of the network interface card adapter card offers two type of optical transceivers; short wavelength laser transceiver and long wavelength laser transceiver.

Both transceivers are in an industry standard 1x9 pinout package.

8.2.1 Short Wavelength Laser Transceiver

- ❏ Duplex SC connectors fully compliant with IEEE 802.3z.
- ❏ Fully compliant with 1000BASE-SX.
- ❏ Supports 50 µm and 62,5 µm multimode fiber
- ❏ Laser class 1 eye safety compliant with 21 CFR(J) and EN60825-1 (+A11).
- ❏ 760 nm (min) to 860 nm (max) wavelength.
- ❏ -9.5 dBm (min) to -3 dBm (max) average launche power.
- ❏ -17 dBm receive sensitivity.
- ❏ 0 dBm (max) average receive power.

8.2.2 Long Wavelength Laser Transceiver

- ❏ Duplex SC connectors fully compliant with IEEE 802.3.z
- ❏ Full compliant with1000BASE-LX
- ❏ Supports 50µm and 62,5µm multimode fiber and 10 µm single mode fiber
- ❏ Laser class 1 eye safety compliant to 21 CFR(J) and EN60825-1 (+A11)
- ❏ 1270 nm (min) to 1355 nm (max) wavelength
- ❏ -11.5dBm (min. MMF) / -11.0 dBm (min. SMF) to -3 dBm (max) average launche power.
- ❏ -19 dBm receive sensitivity.
- ❏ -3 dBm (max) average receive power.

9.0 Technical Data

The table below lists some important technical data describing the "**SK-NET GENESIS <x>**" network interface card.

Bus Interface	64 bit - 66 MHz PCI Local Bus interface achieving a max. transfer rate of 528MByte/s Can also be operated in 32 bit / 33 MHz slots and in combinations e.g. 64 bit / 33 MHz slots. Burst capable bus master Universal I/O (3.3 and 5V) Power Supply 5 V PCI Hot-Plug
Network Interface	Compatible with IEEE 802.3z VLAN -capable Full- /halfduplex operation - auto-negotiation Option: fault tolerant Dual Mac Version
LAN Controller	XMACII from XaQti Corporation
RAM	512 KBytes / 1 MByte / 2 MBytes synchronous SRAMs
FLASH Memory	128 KBytes (Expansion Boot ROM)
EEPROM	512 Bytes for PCI VPD Data
Shared Memory Mapping	16KBytes, including all programmable resources
I/O Address Mapping	256 Bytes Block offers access to the 16 KBytes I/O space via Register Address Port (RAP)
Interrupts	INTA# (on-board interrupt moderation)
Timer	Programmable timer 18.825 nsec resolution
Interrupt Moderation Timer	Programmable timer 18.825 nsec resolution
TCP checksum	On-board generation/checking
Parity	Multiple parity checking/generation points on data path
Power Management	Advanced Power Management supporting D0 and D3 modes

Table 19: Technical Data

Power Dissipation	single MAC: @5V max. 2.6 A (1.6 A typical) dual MAC: @5V max. 3.4 A (2.5 A typical)
Sensors	Temperature sensor and voltage sensor at an I^2C bus
Dimensions	Dimensions are those of a PCI full length add-in board. Including bracket dimensions and retainer: Dual MAC: 12.7 cm × 35.2 cm 5.0 inch × 13.7 inch Single MAC: 12.7 cm × 18.9cm 5.0 inch × 7.4 inch
MTBF	> 500,000 h

Table 19: Technical Data

Operation	10° C — 40° C dry bulb temperature 10 % - 80 % relative humidity max. 27° C wet bulb temperature
Power Off	10° C — 52° C dry bulb temperature 10 % - 80 % relative humidity max. 27° C wet bulb temperature
Storage	1° C — 60° C dry bulb temperature 10 % - 80 % relative humidity max. 29° C wet bulb temperature
Shipment	-20° C — 60° C dry bulb temperature 10 % - 80 % relative humidity max. 29° C wet bulb temperature
Altitude 0 - 3000 feet	10° C — 35° air temperature
Altitude 3000 - 5000 feet	10° C — 32° air temperature

Table 20: Environment Conditions

9.1 Standards Compliance

For functionality “**SK-NET GENESIS <x>**” complies with the international/industry standards — e.g. the IEEE 802.3 standards - ISO standards, and the PCI Local Bus specifications series. Detailed references are listed within this manual or in the corresponding manuals of e.g. the chipset manufacturers.

Additionally, the “**SK-NET GENESIS <x>**” network interface card is designed to meet the following standards and specifications for safety, Electro-Magnetic Compatibility (EMC), ...

Safety
EN60950 - IEC 60950 - VDE 0805
UL 1950
CSA C22.2
CB Certification
EMC:
EN 55022 **IEC - CISPR-22 Class B** (for the fiber versions)
EN 50082-1
EN61000-4-2 **EN61000-4-3** **EN61000-4-4** **EN61000-4-5** **EN61000-4-6** **EN61000-4-8** **EN61000-4-11**
FCC Class B

Table 21: EMC, Safety - Standards Compliance

Note: The ultimate reference for EMC and safety compliance is the CE Declaration of Conformity.

Appendix A
PCI Configuration Register File and Vital Product Data

10.0 PCI Configuration Register File

Providing configuration information and supporting access to configuration information is mandatory for any PCI adapter. These data is available in the PCI configuration register file. The Vital Product Data (VPD) implemented in the "**SK-NET GENESIS <x>**" are an optional PCI extension attached to the configuration register file. The implementation follows an Engineering Change Request (ECR) to the PCI specification version 2.1. Together with PCI Power Management information Vital Product Data are the PCI "New Capabilities" implemented on this adapter.

The PCI configuration register file holds information about the PCI adapter card for seamless integration into the PCI bus system. The file holds data to identify the PCI adapter, about the I/O and memory requirements and about other system resources needed, e.g. interrupt lines, maximum power consumption, etc.

For read / write accesses the configuration space is physically located in the ASIC. Its default /start values are loaded from the Flash EPROM by the Flash EPROM Loader during startup. There's one exception: the Vital Product Data VPD are located in an I^2C EEPROM as required by the PCI ECR. They are read via an Register Address Port (RAP) in the PCI configuration register file.

The configuration registers are set to their default values by **RST#**. Reloading out of the Flash EPROM is initiated with the de-assertion of **RST#** (see Flash EPROM Loader in xxxx).

The "**SK-NET GENESIS <x>**" supports the 256-byte configuration space as defined by the PCI specification revision 2.1.

10.1 Configuration Data Access

The configuration space of a PCI adapter is usually accessed using BIOS routines as described in the PCI BIOS specification. The configuration space of e.g. a network interface card is addressed by selecting the IDSEL (detected by reading the unique DeviceID) + an address in the 256 bytes configuration space address range. This translates to Configuration Read / Configuration Write cycles executed on the PCI bus hardware level.
The "**SK-NET GENESIS <x>**" is responding to type 0 configuration accesses, i.e. AD<1..0> = "00", IDSEL# asserted.

The configuration register file can be accessed with 8-bit, 16-bit or 32-bit transfers. On read transactions all data is driven as defined for full 32-bit accesses independent of BE<3:0>#. All multi-byte numeric fields follow **little-endian** order, if accessed by PCI configuration cycles.

Write operations to reserved or not implemented registers are completed normally on the bus and the data discarded. If the configuration space is targeted for a burst operation, it responds with a disconnect with the first data transfer.

Configuration transactions are not aborted (Target Initiated Termination). Read operations to reserved or not implemented registers are completed normally on the bus and a data value of 0 is returned.

The configuration space of the "**SK-NET GENESIS <x>**" can be accessed also via I/O or memory accesses. For this the configuration register file is mapped into the control register file. This mapping is for test and diagnosis purposes only - it is not recommended for access by driver software.

Each register or portion of a register, which has no fixed value, may be written in test mode (**EN Config Write** is set) with accesses to the **Control Register File** (even, if it is not writeable in the configuration space)**.**

10.2 Overview/Address Map

The table below depicts the layout of the configuration space. Besides the mandatory configuration information in the configuration space header, the "**SK-NET GENESIS <x>**" provides access to two 32-bit registers called "Our_Register 1" and "Our_Register 2" in the device dependent region of the NIC's configuration space. These registers contain information that is important for initialization and not for the "run-time" driver tasks.

The following table shows the address map of the configuration space in a 32-bit (maximum access width for configuration data) register representation. PM is an abbreviation for Power Management.

Header Portion				
Device ID		**Vendor ID**		0x00
Status		**Command**		0x04
Class Code			**Revision ID**	0x08
BIST	**Header Type**	**Latency Timer**	**Cache Line Size**	0x0c
Base Address (1st)				0x10
Base Address (2nd)				0x14
Reserved				0x18
Reserved				0x1c
Reserved				0x20
Reserved				0x24
Reserved				0x28
Subsystem ID		**Subsystem Vendor ID**		0x2c
Expansion Rom Base Address				0x30
Reserved			**Capabilities Ptr**	0x34
Reserved				0x38
Max_Lat	**Min_Gnt**	**Interrupt Pin**	**Interrupt Line**	0x3c
Device Dependent Region				
Our_Register 1				0x40
Our_Register 2				0x44
PM Capabilities		**Next Item Ptr**	**PM Cap_ID**	0x48
PM Data Reg	Reserved	**PM Control/Status**		0x4c
VPD Address Register		**Next Item Ptr**	**VPD Cap_ID**	0x50
VPD Data Register				0x54
Reserved				0x58 ... 0xfc

10.3 Registers

The following tables show the registers in detail on a bit level, list the default values and mentions peculiarities. As stated above configuration space registers are accessed with 8-bit, 16-bit, or 32-bit accesses.

10.3.1 Vendor ID Register

The Vendor ID register comprises 16 bits at address 0x01, 0x00

Bit	Name	Description	Write	Read	Reset value
	Vendor ID Register				
15..0		Identifies SysKonnect as manufacturer of the network interface card.	ne	0x1148	0x1148

The PCI SIG has allocated 0x1148 to SysKonnect as a unique identifier. This value can be reloaded out of the Flash EPROM.

10.3.2 Device ID Register

The Device ID register comprises 16 bits at address 0x03,0x02 that uniquely identifies the network interface card within SysKonnect's product line.

It is reloadable out of the Flash EPROM.

Bit	Name	Description	Write	Read	Reset value
	Device ID Register				
15..0		Identifies the network interface card within SysKonnect's product line.	ne	0x4300	0x4300

10.3.3 Subsystem Vendor ID Register

The Subsystem Vendor ID register comprises 16 bits at address 0x2D, 0x2C. It may be used for customizing OEM versions. The unique Vendor ID of the OEM allocated by the PCI SIG may be provided here.

It is reloadable out of the Flash EPROM.

Bit	Name	Description	Write	Read	Reset value
	Subsystem Vendor ID Register				
15..0		Identifies the subsystem vendor. Must be a valid non-zero value.	ne	0x1148	0x1148

10.3.4 Subsystem ID Register

The Subsystem ID register comprises 16 bits at address 0x2F, 0x2E. It may be used for customizing OEM versions.

It is reloadable out of the Flash EPROM.

Bit	Name	Description	Write	Read	Reset value
15..0		Identifies the subsystem. Must be a valid non-zero value.	ne	0x4300	0x4300

For SysKonnect NICs (not necessarily for OEM versions) the subsytem ID register is foreseen to distinguish various manufacturing options.

10.3.5 Command Register

The command register comprises 16 bits at addresses 0x05,0x04. It is used to control the overall functionality of the network interface card. It controls the network interface card's ability to generate and response to PCI bus cycles.

To disconnect the network interface card logically from all PCI bus cycles except configuration cycles, a value of ZERO should be written to this register.

All bits are reloadable out of the Flash EPROM, except for fixed value bits.

Bit	Name	Description	Write	Read	Reset value
	Subsystem Vendor ID Register				
15 ..10	**Reserved**				
9	**FBTEN**	Fast Back-to-Back enable. If =1, fast back-to-back transactions to different agents are allowed (network interface card as master is running fast back-to-back write cycles) If =0, fast back-to-back transactions are only allowed to the same agent (network interface card as master is not running fast back-to-back write cycles)	yes	aw	0
8	**SERREN**	**SERR#** enable, controls the assertion of **SERR#** pin. If =1, **SERR**# is enabled If =0, **SERR**# is disabled	yes	aw	0

Bit	Name	Description	Write	Read	Reset value
7	**ADSTEP**	Address/data stepping. Fixed value. The "**SK-NET GENESIS <x>**" does not use address/data stepping.	ne	0	
6	**PERREN**	Parity Report Response Enable. If =1, Parity error reporting is enabled	yes	aw	0
5	**VGASNOOP**	VGA palette snoop. Fixed value.	ne	0	
4	**MWIEN**	Memory Write and Invalidate Cycle enable If = 1, Memory Write and Invalidate Cycle is enabled, if = 0, Memory Write must be used instead.	yes	aw	0
3	**SCYCEN**	Special Cycle enable. Fixed value. "**SK-NET GENESIS <x>**" ignores all special Cycle operations.	ne	0	
2	**BMEN**	Bus Master enable If = 1, bus master accesses are enabled, if = 0, bus master accesses are disabled.	yes	aw	0
1	**MEMEN**	Memory Space access enable If = 1, memory accesses are responded, if = 0, memory accesses are not responded.	yes	aw	0
0	**IOEN**	I/O Space access enable If = 1, I/O accesses are responded, if = 0, I/O accesses are not responded.	yes	aw	0

10.3.6 Status Register

The Status Register comprises 16 bits at addresses 0x07, 0x06. It contains status information for the PCI bus related events.

Reads to this register behave normally, writes are slightly different in that bits can be reset, but not set. A bit is reset whenever the register is written, and the data in the corresponding bit location is a ONE. This behavior is marked with ('sh' = special handling) in the table below.

All bits are reloadable out of the Flash EPROM, except for fixed value bits.

Bit	Name	Description	Write	Read	Reset value
Status Register					
15	**PERR**	Parity Error. Is set whenever a parity error is detected (data or address), even if parity error handling is disabled (**PERREN**).	sh	value	0
14	**SERR**	Signaled **SERR#**. Is set whenever an address parity error is detected and both, **SERREN** and **PERREN** are enabled.	sh	value	0
13	**RMABORT**	Received Master Abort. Is set when a master transaction is terminated with a master abort sequence.	sh	value	0
12	**RTABORT**	Received Target Abort. Is set when a network interface card's master transaction is terminated with a Target Abort sequence.	sh	value	0
11	**Reserved**				
10..9	**DEVSEL**	**DEVSEL#** Timing. Fixed value = 01b (medium), **DEVSEL#** is asserted two CLK periods after **FRAME#** is asserted.	ne	01b	
8	**DATAPERR**	Data Parity Error detected. Set, if a data parity error is detected running master cycles and **PERREN** is set.	sh	value	0
7	**FB2BCAP**	Fast Back-to-Back Capable Fixed value = 1, target is capable of accepting fast back-to-back transactions.	ne	1	
6	**UDF**	UDF supported	ne	0	0
5	**66MHZCAP**	66 MHz PCI bus clock capable (see Section 66 MHz Operation xxx)	ne	1	1
4	**NEWCAP**	New capabilities bit = 1 new capabilities list implemented	ne	1	1
3..0	**Reserved**				

10.3.7 Revision ID Register

The Revision ID Register comprises 8-bit at address 0x08. It is reloadable out of the Flash EPROM

Bit	Name	Description	Write	Read	Reset value
	Revision ID Register				
7..0		Specifies the network interface card revision number/Rev. 1.0.	ne	0x10	0x10

10.3.8 Class Code Register

The class code register comprises 24 bits at addresses 0x0B, 0x0A, 0x09. This register is used to identify the generic function of the network interface card. The register is broken in three byte-size fields. One of these fields, the Subclass Register, is reloadable out of the Flash EPROM..

Bit	Name	Description	Write	Read	Reset value
	Programming Interface Register, Lower Byte				
7..0		Specifies the programming interface. Fixed Value = 0	ne	0	0

Bit	Name	Description	Write	Read	Reset value
	Sub-Class Register, Middle Byte				
7..0		Identifies the network controller as an “Ethernet Controller”.	ne	0x00	0x00

Bit	Name	Description	Write	Read	Reset value
	Base-Class Register, Upper Byte				
7..0		Broadly classifies the function of this PCI device as network controller. Fixed Value = 0x02	ne	0x02	0x02

10.3.9 Cache Line Register

The Cache Line Register comprises 8 bits at address 0x0C. The register is reloadable out of the Flash EPROM (not recommended).

Bit	Name	Description	Write	Read	Reset value
	Cache Line Size Register				
7..0		Specifies the system cache line in units of 32-bit words. The "**SK-NET GENESIS <x>**" supports cache line sizes of 2, 4, 8, 16, 32, 64 or 128 32-bit words. The cache line size is restricted to be a power of 2. The most significant 1 in this register is used to set the cache line size. Any other 1's are ignored. A PCI device operated as bus master uses this field as criteria for starting of transfers from/to complete cache lines with the Memory Write and Invalidate, Read Line and Read Multiple commands and to know when to disconnect burst accesses at cache line boundaries.	yes	aw	0

10.3.10 Latency Timer Register

The Latency Timer Register comprises 8 bits at address 0x0D. It is reloadable out of the Flash EPROM (not recommended).

Bit	Name	Description	Write	Read	Reset value
	Latency Timer Register				
7..0		Specifies the maximum time the network interface card can continue with bus master transfers after the system arbiter has removed **GNT#**. The time is specified in units of PCI bus clocks. The working copy of the timer will start counting down when the network interface card asserts FRAME# for the first time during a bus mastership period. The timer will freeze at ZERO. When the timer is ZERO and **GNT#** is de-asserted by the system arbiter, the network interface card will finish the current data phase and then immediately release the bus.	yes	aw	0

10.3.11 Header Type Register

The header type register comprises 8 bits at address 0x0E. This regsiter describes the format of the PCI Configuration space locations 0x10 to 0x3c and states whether the PCI device is a a single function or multi function device.

Bit	Name	Description	Write	Read	Reset value
	Base-Class Register				
7		Single function/multi function device. Fixed value = 0, the network interface card is a single function device.	ne	0	
6..0		PCI configuration space layout. Fixed value = 0, the layout of the PCI configuration space locations 0x10 to 0x3c is as shown in the table above.	ne	0	

10.3.12 Built-in Self Test Register

The optional Built-in Self Test Register comprises 8 bits at address 0x0F.

Bit	Name	Description	Write	Read	Reset value
	Built-in Self Test Register				
7..0		BIST is not supported. Fixed value = 0.	ne	0	

10.3.13 Interrupt Line Register

The Interrupt Line Register comprises 8 bits at address 0x3C.

Bit	Name	Description	Write	Read	Reset value
	Interrupt Line Register				
7..0		The Interrupt Line Register is used to communicate interrupt line routing information. POST software writes the routing information into this register as it initializes and configures the system. The value in this register tells which input of the system interrupt controller(s) the device's interrupt pin is connected to. Device drivers and operating systems can use this information to determine priority and vector information. The interrupt line register is not modified by the network interface card. It has no effect on the operation of the device.	yes	aw	0

10.3.14 Interrupt Pin Register

The Interrupt Pin register comprises 8 bits at address 0x3D.

Bit	Name	Description	Write	Read	Reset value
	Interrupt Pin Register				
7..0		Fixed value, the network interface card is using the interrupt pin **INTA#**. Fixed value.	ne	0x01	

10.3.15 Min_Gnt Register

The Min_GNT register comprises 8 bits at address 0x3E. It is reloadable out of the Flash EPROM.

Bit	Name	Description	Write	Read	Reset value
	Min_Gnt Register				
7..0		This read-only register specifies the network interface card's desired settings for value of the Latency Timer. The value specifies in units of 1/4 microseconds the burst period needed by the network interface card assuming a clock rate of 33MHz. (64 x 64-bit words x 30ns) = 1.92µs 1.92us/0.25µs = 8	ne	0x08	0x08

10.3.16 Max_Lat Register

The Max_Lat register comprises 8 bits at address 0x3F. It is reloadable out of the Flash EPROM.

Bit	Name	Description	Write	Read	Reset value
	Max_Lat Register				
7..0		This read-only register specifies the's desired settings for Latency Timer value. The value specifies in units of 1/4 microseconds how often the network interface card needs to gain access to the PCI bus assuming a clock rate of 33MHz. (64 x 64-bit words x 8)/250MB/s = 2µs 2us/0.25µs = 8	ne	0x08	0x08

10.3.17 Base Address Register (1st)

The 1st Base Address Register is a 32-bit register starting at address 0x10. The register determines the location of the network interface card in memory space, if memory mapping is used.

The base address register is reloadable out of the Flash EPROM

Bit	Name	Description	Write	Read	Reset value
Base Address Register (1st)					
31..14	**MEMBASE**	Memory base address most significant 18 bits.	yes	aw	0
13..4	**MEMSIZE**	Memory size requirements. Fixed value, indicates memory space requirement of 16384 bytes.	ne	0	
3	**PREFEN**	Prefetchable. Fixed value, indicates that prefetching is not allowed. (Memory Write Byte Merging is not tolerable for this PCI device.)	ne	0	
2..1	**Memory Type**	Memory type. Indicates, that base register is 32 bits wide, and mapping can be done anywhere in the 32-bit memory space. **Memory Type** may be reloaded from the Flash EPROM (further memory types).	ne	0	0
0	**MEMSPACE**	Memory space indicator. Fixed value, indicates, that this **Base Address Register** describes a memory base address.	ne	0	

10.3.18 Base Address Register (2nd)

The 2nd Base Address Register is a 32 bit register starting at address 0x14. The register determines the location of the network interface card in all of I/O space.

The register is reloadable out of the Flash EPROM. If **EN IO Mapping** is disabled, this location is treated like **reserved** locations.

Bit	Name	Description	Write	Read	Reset value
	Base Address Register (2nd)				
31..8	**IOBASE**	I/O base address most significant 24 bits.	yes	aw	0
7..2	**IOSIZE**	I/O size requirements. Fixed value. Reading back a value of “0” in bits 7..2 indicates I/O space requirement of 256 bytes.	ne	0	
1	**Reserved**				
0	**IOSPACE**	I/O space indicator. Indicating that this Base Address Register describes an I/O base address.	ne	1	1

10.3.19 Expansion Rom Base Address Register

The Expansion Rom Base Address register is a 32-bit register starting at address 0x30 that determines the base address and size information of the Expansion Rom. The Expansion ROM Base Address register is reloadable out of the Flash EPROM.

If **EN EPROM** is disabled, this location is treated like **reserved** locations.

Bit	Name	Description	Write	Read	Reset value
	Expansion Rom Base Address Register				
31..17	**ROMbase**	ROM base address significant 15 bits.	yes	aw	0
16..14	**ROMbase/size**	Treated as **ROMBASE** or **ROMSIZE** depending on settings of **Pagesize**.	yes ne	aw 0	0

Bit	Name	Description	Write	Read	Reset value
13..11	**ROMsize**	ROM size requirements. Fixed value. Reading back a value of “0” indicates memory space requirement of 16k bytes or higher.	ne	0	
10..1	**Reserved**				
0	**ROMEN**	Address decode enable. Read/write accessible. If ROMEN = 0, the devices Expansion ROM address space is disabled. If ROMEN = 1 and MEMEN =1 (Command Register), the devices Expansion ROM address space is enabled.	yes	aw	0

10.3.20 New Capabilities Pointer

The New Capabilities Pointer Register is a 8-bit register at address 0x34 that points to the New Capabilities List.

This pointer register is reloadable out of the Flash EPROM.

Bit	Name	Description	Write	Read	Reset value
New Capabilities Pointer Register					
7..0	**New Capabilities Pointer**	Points to the New Capabilities list	ne	0x48	0x48

10.4 Device Specific Region

Our registers are the first two used locations in the “device specific region” of the 256-byte configuration space. Because there is no reliable procedure for setting these registers at power-on, the default values are chosen for the most common environments.

Modifications may be handled as manufacturing option, driver options, dedicated configuration software (SK, reprogramming the Flash EPROM) or the UDF mechanism, where available (‘User Definable Features’ using a PCI Configuration File (PCF) as introduced, but not required by PCI Specification/Rev. 2.1).

This may be used as manufacturing option, if it seems practically supporting the network interface card with a PCI Configuration File (PCF) for configuring **Our Register 1** and **2** (PCI Specification/Rev. 2.1).

10.4.1 Our Register 1 , 2

Our Register 1 is a 32-bit register at address 0x40. Our Register 2 is a 32-bit register at address 0x44. Both are reloadable out of the Flash EPROM.
Most of the switches in **Our Register 1** and **2** are not intended for use at run time. Manufactured adapters may come up with different settings than defined as **Reset value** (if reloaded out of the Flash EPROM).

The fields marked in column **write** with “ne” are writeable only in test-mode. The fields marked with “yes” are writeable in configuration space and with normal accesses to the **Control Register File**.

Bit	Name	Description	Write	Read	Reset value
Our Register 1					
31..26	Reserved				
25	**VIO**	Voltage for PCI I/O buffers. 0 = 3.3V 1 = 5.0V	ne	value	
24	**EN Boot**	Boot enable, for software purposes only if EN_BOOT = 0, boot with expansion ROM code if **EN Boot** = 1, don’t boot with expansion ROM code.	yes	aw	0
23	**EN IO Mapping**	Controls mapping of the **Control Register File** to the I/O space (manufacturing option). If disabled (**EN IO Mapping** = 0), any address decoding for I/O accesses is disabled (see also **Base Address Register (2nd)**).	ne	1	1
22	**EN EPROM**	Controls mapping of the Flash EPROM to the memory space. If disabled (**EN EPROM** = 0), any address decoding for memory accesses is disabled (see also **ROM Base Address Register**).	ne	1	1

Bit	Name	Description	Write	Read	Reset value
21..20	**Pagesize<1..0>**	Pagesize/Flash EPROM Defines the size, which is mapped to the system (see **ROMsize/ROMbase**). 3 = 128k 2 = 64k 1 = 32k 0 = 16k	ne	3	3
19	Reserved				
18 ..16	**Page Reg<2..0>**	Page Register Selects the page of the Flash EPROM space, which is mapped to the system, if **Pagesize** is lower than 128k. 0 = page 0, 1 = page 1 ... 7 = page 7 **May be undefined on booting without hardware reset.**	yes	aw	0
15	**No Tar**	No turnaround cycle on external RAMs 0 = default: turnaround cycle	yes	aw	0
14	**Force_BE**	Assert all BE<7..0># on Master-Reads	yes	aw	0
13	**Dis MRL**	Disable command **Memory Read Line**	yes	aw	0
12	**Dis MRM**	Disable command **Memory Read Multiple**	yes	aw	0
11	**Dis MWI**	Disable command **Memory Write and Invalidate**	yes	aw	0
10	**Disconnect at CLS**	Disconnect at Cache Line Size Boundary (command **Memory Write And Invalidate**)	yes	aw	0
9	**Burst Disable**	Burst disable. Disables bursting on master transfers as a patch for systems not supporting burst transfers.	yes	aw	0
8	**Dis PCI Clock**	Disable driving the PCI clock onto the board 1= disable 0= enable (Default MUST be “enable” guaranteeing clock at POWER ON RESET)	ne	value	0
7..4	**Skew/PCI<3..0>**	Skew Control, PCI clock on board	ne	value	to be determined by simulation

Bit	Name	Description	Write	Read	Reset value
3..0	**Skew/Host<3..0>**	Skew Control, internal host clock tree	ne	value	to be determined by simulation
Our Register 2 (0x44)					
31..24	**VPD Write Thr**	Defines the first address of the writeable VPD area in steps of 128 Bytes. Default value is 128 Bytes = address 0x80. For manufacturing programming of the VPD EEPROM it must be set to 0 in testmode.	ne	0x01	0x01
23..17	**VPD Devsel**	Defines the Device Select Byte for the serial EEPROM used for VPD storage. Default value is 0b1010000.	ne	0x50	0x50
16..14	**VPD ROM Size**	Defines the size of the assembled serial EEPROM in Bytes. 0 = 256 Bytes 1 = 512 Bytes 2 = 1024 Bytes 3 = 2048 Bytes 4 = 4096 Bytes 5 = 8192 Bytes 6 = 16384 Bytes 7 = 32768 Bytes Default value is 256 Bytes. If any other size is used, this field must be reprogrammed out of the Flash EPROM.	ne	0x00	0x00
13..12	Reserved				
11..8	**Patch Dir<3..0>**	Defines the type of the pins External Patchs. 1 = output 0 = input	yes	aw	0
7..4	**Ext Patchs<3..0>**	These bits are routed to the ASIC's pins for future external configuration options.	yes	aw	0
3	**En Dummy Read**	Enable dummy read for put descriptor operations: If **En Dummy Read** = 1, a dummy read cycle is inserted before writing Receive/Transmit Buffer Control (OWN), if there was a descriptor write before	yes	aw	1

Bit	Name	Description	Write	Read	Reset value
2	**Rev_Bytes_Desc**	Revert byte order in any descriptor's 32-bit words. Changes byte ordering for descriptor 64-bit words for use in big endian systems. If **Rev_Bytes_Desc** = 0, AD<31..24> / AD<23..16> / AD<15..8> / AD<7..0> are stored in Bit31..24 / Bit23..16 / Bit15..8 / Bit7..0 of the Descriptor Registers. If **Rev_Bytes_Desc** = 1, AD<31..24> / AD<23..16> / AD<15..8> / AD<7..0> are stored in Bit7..0 / Bit15..8 / Bit23..16 / Bit31..24 of the descriptor registers.	yes	aw	0
1	Reserved				
0	**UseData64**	Controls usage of the 64-bit data bus extension in Bus master mode: If **UseData64**=0, the **REQ64#** line is never asserted by the network interface card. If **UseData64**=1, the **REQ64#** line is asserted by the network interface card for all busmaster transfers, except for all descriptor operations.	yes	aw	1

Skew Bits

The **Skew/Host<3..0>** and the **Skew/PCI<3..0>** were introduced to correct skew problems at critical points in the clock distribution. Skew/Host<3..0> adds a delay of 0.5 nsec/bit, Skew/PCI<3..0> adds 0.2 nsec/bit. The Skew/Host<3..0> were foreseen to shift the PCI clock available in the adapter's clock distribution tree. However, since the PCI clock is not used on the current version of the "**SK-NET GENESIS <x>**" Skew/Host<3..0> has no effect.

Skew/PCI<3..0> controlls a clock delay between the ASIC and the XMACII Gigabit Ethernet Controller. The setting of Skew/Host<3..0> and Skew/PCI<3..0> is design dependent and has been found by timing analysis for the "**SK-NET GENESIS <x>**".

☞ **Skew bits are critical for correct operation of the network interface card. They must be defined by the card's designers and must not be changed.**

10.4.2 Power Management Capability ID Register

The Power Management Capabilty ID comprises 8 bits at address 0x48 . Power Management is the first "New Capability" in the New Capability list. The New Capabilites pointer at address 0x34 contains the address 0x48 for the first entry in the New Capabilities list. The Next Item Pointer at address 0x49 holds the address of the next entry in the New Capability list. For the "**SK-NET GENESIS <x>**" this is the Vital Product Data VPD New Capability ID. The attached Next Item Pointer holds a 0x0 in the "**SK-NET GENESIS <x>**" indicating the end of the New Capabilities list.
The New Capabilities linked list uses New Capability IDs for the unique

identification of the capabilty. New Capabiliy IDs are assigned by the PCI SIG. The structure of New Capability information is also specified by the PCI SIG. E.g. Power Management has a New Capability ID of 0x01. The structure of the related information is specified in the PCI Power Management Interface Specification. The ID for VPD is 0x03. The structure of New Capability information is defined in the ECR to PCI specification 2.1 and will probably be integrated in later specification versions.

10.4.3 Power Management Capability ID Register

The Power Management New Capability ID comprises 8 bit at 0x48. It is reloadable out of the Flash EPROM.

Bit	Name	Description	Write	Read	Reset value
	Power Management Capability ID Register				
7..0	**Cap_ID**	Power Management Capabilities ID	ne	0x01	0x01

10.4.4 Power Management Next Item Pointer

The Power Management Next Item Pointer comprises 8 bits at address 0x49. It is reloadable out of the Flash EPROM.

Bit	Name	Description	Write	Read	Reset value
	Power Management Next Item Pointer				
7..0	**Next Item Ptr**	Pointer to the next item in the capabilities list	ne	0x50	0x50

10.4.5 Power Management Capabilities Register

The Power Management Capabilities Register comprises 16 bit at addresses 0x4B, 0x4A. It is reloadable out of the Flash EPROM.

Bit	Name	Description	Write	Read	Reset value
	Power Management Capabilities Register				
15..11	**PME_Support**	Power Management Event Support: Specifies in which power state the Signal PME# may be asserted. The "**SK-NET GENESIS <x>**" doesn't support PME#.	ne	0x00	0x00
10	**D2_Support**	D2 Support: The "**SK-NET GENESIS <x>**" doesn't support D2 Power Management State.	ne	0	0
9	**D1_Support**	D1 Support: The "**SK-NET GENESIS <x>**" doesn't support D1 Power Management State.	ne	0	0

Bit	Name	Description	Write	Read	Reset value
8..6	**Reserved**	Reserved, but reloadable out of the Flash EPROM for changes in the PCI Specification.	ne	0x00	0x00
5	**DSI**	Device specific initialization: The "**SK-NET GENESIS <x>**" requires device specific initialization	ne	1	1
4	**APS**	Auxiliary Power Source: The "**SK-NET GENESIS <x>**" requires no auxiliary power source, as it doesn't support PME#	ne	0	0
3	**PME Clock**	Power Management Event Clock: The "**SK-NET GENESIS <x>**" doesn't need PCI clock, as it doesn't support PME#	ne	0	0
2:0	**Version**	The "**SK-NET GENESIS <x>**" complies with Revision 1.0 of the PCI Power Management Interface Specification	ne	0x01	0x01

10.4.6 Power Management Control/Status Register

The Power Management Control/Status register comprises 16 bit at addresses 0x4C. It is reloadable out of the Flash EPROM.

Bit	Name	Description	Write	Read	Reset value
	Power Management Control/Status Register				
15	**PME_Status**	The "**SK-NET GENESIS <x>**" doesn't support PME#	ne	0	0
14..13	**Data_Scale**	Indicates the scaling factor to be used when interpreting the value of the Data Register. The read value depends on the setting of the **Data_Select** field.	ne	value	0b01
12..9	**Data_Select**	This 4-bit field is used to select which data is to be reported through the Data Register and **Data_Scale** field.	yes	aw	0

Bit	Name	Description	Write	Read	Reset value
8	**PME_En**	The "**SK-NET GENESIS <x>**" doesn't support PME# generation from D3 cold.	ne	0	0
7..2	**Reserved**				
1:0	**Power State**	Controls the Power Management State of the network interface card. If written with an unsupported value (1 = D1, 2 = D2), the write operation is completed normally on the bus, but the write data is discarded and no state change occurs.	yes, only values 0 and 3	aw	0

The 16 2-bit **Data_Scale** fields and the 16 8-bit **Data** register values, that can be selected by the **Data_Select** field, are reloadable out of the Flash EPROM by writing complete 32-bit wide sets of (**Data_Select**, **Data_Scale**, **Data**) to the **Power Management Control/Status** and **Data Register**.

☞ **Warning: To modify the contents of any Data_Scale or Data field the Data_Select field MUST always be written with the desired Data_Select value in the same 32-bit access!**

10.4.7 Power Management Data Register

The Power Management Data register comprises 8 bits at address 0x4F. It is reloadable out of the Flash EPROM.

Bit	Name	Description	Write	Read	Reset value
	Power Management Data Register				
7..0	**Data**	This read-only register is used to report the state dependent data requested by the **Data_Select** field. The value of this register is scaled by the value reported by the **Data_Scale** field.	ne	value	0x19

10.4.8 Power Management Data Table

Data and **Data_Scale** are hidden register accessible through the **Power Management Registers.**

Data_Scale is writeable by the Flush EPROM loader by writing to **Power Management Control/Status Register.**

Data is writeable by the Flush EPROM loader by writing to **Power Management Data Register.**

Data and **Data_Scale** are reloaded from the Flash EPROM with values matching the manufacturing option. See "Flash EPROM Reloads".

The two version of the "**SK-NET GENESIS <x>**" may differ in their power consumption. A separate Power management Data Table must be assigned to each version separately.

Single MAC version:

Value in Data_Select	Meaning	Data (8 bit) Reset Value	Data_Scale (2 bit) Reset Value	[Watt]
0	**D0** Power consumed	0x90	0b01	14.4
1	**D1** Power consumed	0	0	na
2	**D2** Power consumed	0	0	na
3	**D3** Power consumed	0x3D	0b01	6.1
4	**D0** Power dissipated	0x90	0b01	14.4
5	**D1** Power dissipated	0	0	na
6	**D2** Power dissipated	0	0	na
7	**D3** Power dissipated	0x3D	0b01	6.1
8..15	For multifunction devices only	0	0	0

Dual MAC version:

Value in Data_Select	Meaning	Data (8 bit) Reset Value	Data_Scale (2 bit) Reset Value	[Watt]
0	**D0** Power consumed	0xBC	0b01	18.8
1	**D1** Power consumed	0	0	na
2	**D2** Power consumed	0	0	na
3	**D3** Power consumed	0x5F	0b01	9.5
4	**D0** Power dissipated	0xBC	0b01	18.8
5	**D1** Power dissipated	0	0	na
6	**D2** Power dissipated	0	0	na
7	**D3** Power dissipated	0x5F	0b01	9.5
8..15	For multifunction devices only	0	0	0

10.4.9 VPD Capability ID Register

The VPD Capability ID register comprises 8 bits at address 0x50. The register is reloadable out of the Flash EPROM. It contains the New Capability ID for Vital Product Data (VPD) as specified by the PCI SIG.

Bit	Name	Description	Write	Read	Reset value
	VPD Capability ID Register				
7..0	**Cap_ID**	VPD Capabilities ID	ne	0x03	0x03

10.4.10 VPD Next Item Pointer

The VPD Next Iterm Pointer comprises 8 bits at address 0x51. It is reloadable out of the Flash EPROM.

Bit	Name	Description	Write	Read	Reset value
	VPD Next Item Pointer				
7..0	**Next Item Ptr**	Pointer to the next item in the capabilities list	ne	0x00	0x00

10.4.11 VPD Address Register

The VPD Address Register comprises 16 bits at address 0x52. It is reloadable out of the Flash EPROM. The VPD address register is also writeable in I/O space.

The **VPD Address & Data Registers** are controlling an I^2C interface, which is running a 100 kHz protocol to an external I^2C EEPROM.

The interface signals are routed through the pins VPD_CLOCK and VPD_DATA.

The I^2C clock and data port pins are pulled high by a pull up resistor to VCC of the I^2C device.

Bit	Name	Description	Write	Read	Reset value
	VPD Address Register				
15	**Flag**	Starts the VPD data transfers, determines its direction and signals its completion by being toggled by H/W. If written 1, a VPD write is started, will be set to 0 after completion. If written 0, a VPD read is started, will be set to 1 after completion.	exec	value	0
14..0	**VPD Address**	Address of the VPD contents to be written / read.	yes	aw	0x00

10.4.12 VPD Data Register

The VPD data register is a 32-bit register at address 0x54. It is reloadable out of the Flash EPROM and is also writeable in I/O space.

Bit	Name	Description	Write	Read	Reset value
	VPD Data Register				
31:0	**VPD Data**	Must be written before VPD Address Register for VPD write. Contains VPD read Data after completion of VPD read.	yes	aw	0x00

10.5 VPD EEPROM

The network interface card implements VPD as suggested in ECR (03/28/97). It is stored in a serial EEPROM and may be accessed through the **VPD Address Register** and **VPD Data Register**. These registers are mapped writeable both in configuration and I/O address space.

The serial I^2C protocol is handled by the ASIC hardware. The I^2C address of the serial EEPROM for VPD is 1010000.

As serial EEPROM Atmel's AT24C02 or equivalent is initially used.

The size of the serial EEPROM, the amount of writeable VPD area and the Device Select Byte used for VPD I^2C accesses are determined by the read only fields **VPD ROM Size**, **VPD Write Threshold** and **VPD Device Select** in Our Register 2. These fields can be reloaded from the Flash EPROM, if another device is used.

For manufacturing programming of the read only part of VPD, the **VPD Write Threshold** must be set to 0 in Testmode (**Enable Config Write**). Then the whole serial EEPROM is writeable. After the next power cycle the **VPD Write Threshold** will be on it's default setting and the read only area will be write protected.

Appendix B
Register Overview
Control Register File

11.0 Control Register File

The control register file comprises all registers accessible to the host, independent of their specific physical location. This includes the registers in the ASIC but also the XMACII registers and the PCI configuration register file. FPROM data and VPD data are accessible through Register Address Ports (RAP) implemented in the control register file.

The control register file may be mapped in the 32-bit I/O space as well as in the 32-bit memory space depending on **Base Address Register (1st)** and **Base Address Register (2nd).**

If the control register file is mapped into the memory space, it covers a memory range of 16 Kbytes. All registers may be accessed directly.
If the Control Register File is mapped into the I/O space, it's covering an I/O range of 256 bytes. The 16 K range of the control register file is segmented into 128 x 128-byte blocks.

Block 0 is mapped permanently to the lower half of the 256 byte I/O range. (Block 0 holds for example the main control register where the Reset signals are switched and the information which block is mapped to the upper half of the 256 bytes.)
Block 0 - 127 are mapped to the upper half of the 256byte I/O range as defined by the Register Address Port (RAP) in Block 0.[1]

Write operations to reserved or not implemented registers are completed normally on the bus and the data are discarded.
Read operations to reserved or not implemented registers are completed normally on the bus, read data are undefined.

If **SW Reset** is set, it is not recommended to access the **Control Register File** except for accessing the main **Control Register** (all internal or external devices are in reset state).
On **SW Reset** write operations are completed normally on the bus and the data discarded. Read operations are completed normally on the bus and a data value of 0 returned.

In order to prevent setup and hold time violations, all signals and/or events routed through read registers are synchronized to the PCI bus **CLK** signal. All signals and/or events routed through **INTA#** are glitch free and, except for those directly related to **CLK** (PCI) they are independent from **CLK** (PCI).

11.1 Overview/Address Map

The following table depicts the layout of the control register file. The contents of individual registers down to bit level will be shown in the following sections.

1. Note: As a side effect, register addressing with the RAP may also be used, if the control register file is mapped into the memory space

Column ‘STI’ in the following table is marking registers residing behind the Same Target Interface.

Byte<3>	Byte<2>	Byte<1>	Byte<0>	Memory Address	STI	I/O RAP Block
Reserved	Reserved	Reserved	**Register Address Port (RAP)**	0x0000		0
Reserved	**LED Register**	**Control/Status Register**		0x0004		
Interrupt Source Register				0x0008		
Interrupt Mask Register				0x000c		
Interrupt Hardware Error Source Register				0x0010		
Interrupt Hardware Error Mask Register				0x0014		
Special Interrupt Source Register				0x0018		
Reserved				0x001c		
Reserved	Reserved	**XMACII 1: Interrupt Mask Register**		0x0020		
Reserved	Reserved	Reserved		0x0024		
Reserved	Reserved	**XMACII 1: Interrupt Status Register**		0x0028		
Reserved	Reserved	Reserved		0x002c		
Reserved	Reserved	**XMACII 1: PHY Address Register**		0x0030		
Reserved	Reserved	**XMACII 1: PHY Data Register**		0x0034		
Reserved	Reserved	Reserved		0x0038		
Reserved	Reserved	Reserved		0x003c		
Reserved	Reserved	**XMACII 2: Interrupt Mask Register**		0x0040		
Reserved	Reserved	Reserved		0x0044		
Reserved	Reserved	**XMACII 2: Interrupt Status Register**		0x0048		
Reserved	Reserved	Reserved		0x004c		

Byte<3>	Byte<2>	Byte<1>	Byte<0>	Memory Address	STI	I/O RAP Block
Reserved	Reserved	**XMACII 2: PHY Address Register**		0x0050		
Reserved	Reserved	**XMACII 2: PHY Data Register**		0x0054		
Reserved	Reserved	Reserved		0x0058		
Reserved	Reserved	Reserved		0x005c		
BMU Control/Status Register (Receive Queue 1)				0x0060		
BMU Control/Status Register (Receive Queue 2)				0x0064		
BMU Control/Status Register (Sync. Transmit Queue 1)				0x0068		
BMU Control/Status Register (Async. Transmit Queue 1)				0x006c		
BMU Control/Status Register (Sync. Transmit Queue 2)				0x0070		
BMU Control/Status Register (Async. Transmit Queue 2)				0x0074		
Reserved				0x0078		
Reserved				0x007c		
Block Window Register <31:0> Note: If RAP = 1, read values are ZERO, write cycles have no effect				0x0080		1

Byte<3>	Byte<2>	Byte<1>	Byte<0>	Memory Address	STI	I/O RAP Block
MAC Addr_1<3>	**MAC Addr_1<2>**	**MAC Addr_1<1>**	**MAC Addr_1<0>**	0x0100		
Reserved	Reserved	**MAC Addr_1<5>**	**MAC Addr_1<4>**	0x0104		
MAC Addr_2<3>	**MAC Addr_2<2>**	**MAC Addr_2<1>**	**MAC Addr_2<0>**	0x0108		
Reserved	Reserved	**MAC Addr_2<5>**	**MAC Addr_2<4>**	0x010c		
MAC Addr_3<3>	**MAC Addr_3<2>**	**MAC Addr_3<1>**	**MAC Addr_3<0>**	0x0110		
Reserved	Reserved	**MAC Addr_3<5>**	**MAC Addr_3<4>**	0x0114		
Chip Revision	**Configuration**	**PMD Type**	**Connector Type**	0x0118		
Eprom<3>	**Eprom<2>**	**Eprom<1>**	**Eprom<0>**	0x011c		
						2
EPROM Address Register/Counter				0x0120		
Reserved	Reserved	Reserved	**EPROM Data Port**	0x0124		
Reserved	Reserved	**Loader Test**	**Loader Control**	0x0128		
Reserved				0x012c		
Timer Init Value				0x0130		
Timer				0x0134		
Reserved	Reserved	**Timer Test**	**Timer Control**	0x0138		
Reserved				0x013c		
IRQ Moderation Timer Init Value				0x0140		
IRQ Moderation Timer				0x0144		

Byte<3>	Byte<2>	Byte<1>	Byte<0>	Memory Address	STI	I/O RAP Block
Reserved	Reserved	**IRQ Moderation Timer Test**	**IRQ Moderation Timer Control**	0x0148		
Interrupt Moderation Mask Register				0x014c		
Interrupt Hardware Error Moderation Mask Register				0x0150		
Reserved				0x0154		
Reserved		**Test Control Reg 2**	**Test Control Reg 1**	0x0158		
General Purpose I/O Register				0x015c		
I2C (HW) Control Register				0x0160		
I2C (HW) Data Register				0x0164		
I2C(HW) IRQ Register				0x0168		
I2C (SW) Register				0x016c		
Blink Source Counter Init Value				0x0170		
Blink Source Counter				0x0174		
Blink Source Counter Test		**Blink Source Counter Status**	**Blink Source Counter Control**	0x0178		
Reserved				0x017c		

Byte<3>	Byte<2>	Byte<1>	Byte<0>	Memory Address	STI	I/O RAP Block
RAM Random Registers						
Ram Address				0x0180		
Ram Data Port/lower dword				0x0184		3
Ram Data Port/upper dword				0x0188		
Reserved				0x018c		

Byte<3>	Byte<2>	Byte<1>	Byte<0>	Memory Address	STI	I/O RAP Block
RAM Interface Registers						
Timeout Init Value 3 (Read SM Rx1)	**Timeout Init Value 2 (Write SM Tx/s1)**	**Timeout Init Value 1 (Write SM Tx/a1)**	**Timeout Init Value 0 (Write SM Rx1)**	0x0190		
Timeout Init Value 7 (Write SM Tx/a2)	**Timeout Init Value 6 (Write SM Rx2)**	**Timeout Init Value 5 (Read SM Tx/s1)**	**Timeout Init Value 4 (Read SM Tx/a1)**	0x0194		
Timeout Init Value 11 (Read SM Tx/s2)	**Timeout Init Value 10 (Read SM Tx/a2)**	**Timeout Init Value 9 (Read SM Rx2)**	**Timeout Init Value 8 (Write SM Tx/s2)**	0x0198		
Reserved			**Timeout Timer**	0x019c		
Reserved	**Timer Test**	**Ram Interface Control**		0x01a0		
Reserved				0x01a4		
Reserved				0x01a8		
Reserved				0x01ac		
MAC Arbiter Registers						3
Timeout Init Value Tx2	**Timeout Init Value Tx1**	**Timeout Init Value Rx2**	**Timeout Init Value Rx1**	0x01b0		
Timeout Timer Tx2	**Timeout Timer Tx1**	**Timeout Timer Rx2**	**Timeout Timer Rx1**	0x01b4		
Timer Test		**MAC Arbiter Control**		0x01b8		
Reserved				0x01bc		
Recovery Init Value Tx2	**Recovery Init Value Tx1**	**Recovery Init Value Rx2**	**Recovery Init Value Rx1**	0x01c0		
Recovery Timer Tx2	**Recovery Timer Tx1**	**Recovery Timer Rx2**	**Recovery Timer Rx1**	0x01c4		
Timer Test		**Recovery Control**		0x01c8		
Reserved				0x01cc		

Byte<3>	Byte<2>	Byte<1>	Byte<0>	Memory Address	STI	I/O RAP Block
Packet Arbiter Registers						
Reserved		**Timeout Init Value Rx1**		0x01d0		3
Reserved		**Timeout Init Value Rx2**		0x01d4		
Reserved		**Timeout Init Value Tx1**		0x01d8		
Reserved		**Timeout Init Value Tx2**		0x01dc		
Reserved		**Timeout Rx1**		0x01e0		
Reserved		**Timeout Rx2**		0x01e4		
Reserved		**Timeout Tx1**		0x01e8		
Reserved		**Timeout Tx2**		0x01ec		
Timer Test		**Packet Arbiter Control**		0x01f0		
Reserved				0x01f4 : 0x01fc		
Transmit Arbiter MAC 1						4
Interval Timer Init Value				0x0200		
Interval Timer				0x0204		
Limit Counter Init Value				0x0208		
Limit Counter				0x020c		
Reserved	**Timer/ Counter Status**	**Timer/ Counter Test**	**Timer/ Counter Control**	0x0210		
Reserved				0x0214 : 0x021c		
Reserved				0x0220 : 0x027c		

Byte<3>	Byte<2>	Byte<1>	Byte<0>	Memory Address	STI	I/O RAP Block
Transmit Arbiter MAC 2						
Interval Timer Init Value				0x0280		
Interval Timer				0x0284		
Limit Counter Init Value				0x0288		
Limit Counter				0x028c		5
Reserved	**Timer/ Counter Status**	**Timer/ Counter Test**	**Timer/ Counter Control**	0x0290		
Reserved				0x0294 : 0x029c		
Reserved				0x02a0 : 0x02fc		
Reserved	Reserved	**External Register <0x00> . . External Register <0x1f>**		0x0300 . . 0x037c		6
Configuration Register File (lower half)				0x0380 : 0x03fc		7

Byte<3>	Byte<2>	Byte<1>	Byte<0>	Memory Address	STI	I/O RAP Block
Receive Queue 1						8
Current Receive Descriptor						
Receive Buffer Control		**Receive Buffer Byte Count**		0x0400		
Next Receive Descriptor Address, lower dword				0x0404		
Receive Buffer Address, lower dword				0x0408		
Receive Buffer Address, upper dword				0x040c		
Receive Frame Status Word				0x0410		
Time Stamp				0x0414		
TCP/IP checksum extension						
Checksum 2		**Checksum 1**		0x0418		
Start position, Checksum 2		**Start position, Checksum 1**		0x041c		
Current Receive Descriptor Address, lower dword				0x0420		
Current/Next Receive Descriptor Address, upper dword				0x0424		
Current Address Counter, lower dword				0x0428		
Current Address Counter, upper dword				0x042c		
Current Byte Counter				0x0430		
BMU Control/Status Register				0x0434		
Flag Register including FIFO Address Counters				0x0438		
Test Register 1				0x043c		
Test Register 2				0x0440		
Test Register 3				0x0444		
Reserved				0x0448 : 0x047c		

Byte<3>	Byte<2>	Byte<1>	Byte<0>	Memory Address	STI	I/O RAP Block
Receive Queue 2						9
Current Receive Descriptor						
Receive Buffer Control		**Receive Buffer Byte Count**		0x0480		
Next Receive Descriptor Address, lower dword				0x0484		
Receive Buffer Address, lower dword				0x0488		
Receive Buffer Address, upper dword				0x048c		
Receive Frame Status Word				0x0490		
Time Stamp				0x0494		
TCP/IP checksum extension						
Checksum 2		**Checksum 1**		0x0498		
Start position, Checksum 2		**Start position, Checksum 1**		0x049c		
Current Receive Descriptor Address, lower dword				0x04a0		
Current/Next Receive Descriptor Address, upper dword				0x04a4		
Current Address Counter, lower dword				0x04a8		
Current Address Counter, upper dword				0x04ac		
Current Byte Counter				0x04b0		
BMU Control/Status Register				0x04b4		
Flag Register including FIFO Address Counters				0x04b8		
Test Register 1				0x04bc		
Test Register 2				0x04c0		
Test Register 3				0x04c4		
Reserved				0x04c8 : 0x04fc		
Reserved	Reserved	Reserved		0x0500 **:** 0x05fc		10 : 11

Byte<3>	Byte<2>	Byte<1>	Byte<0>	Memory Address	STI	I/O RAP Block
Sync. Transmit Queue 1						12
Current Transmit Descriptor						
Transmit Buffer Control		**Transmit Buffer Byte Count**		0x0600		
Next Transmit Descriptor Address, lower dword				0x0604		
Transmit Buffer Address, lower dword				0x0608		
Transmit Buffer Address, upper dword				0x060c		
Transmit Frame Status Word				0x0610		
TCP/IP checksum extension						
Reserved		**Start value**		0x0614		
Start position		**Write position**		0x0618		
Reserved				0x061c		
Current Transmit Descriptor Address, lower dword				0x0620		
Current/Next Transmit Descriptor Address, upper dword				0x0624		
Current Address Counter, lower dword				0x0628		
Current Address Counter, upper dword				0x062c		
Current Byte Counter				0x0630		
BMU Control/Status Register				0x0634		
Flag Register including FIFO Address Counters				0x0638		
Test Register 1				0x063c		
Test Register 2				0x0640		
Test Register 3				0x0644		
Reserved				0x0648 : 0x067c		

Byte<3>	Byte<2>	Byte<1>	Byte<0>	Memory Address	STI	I/O RAP Block
Async. Transmit Queue 1						
Current Transmit Descriptor						
Transmit Buffer Control		**Transmit Buffer Byte Count**		0x0680		
Next Transmit Descriptor Address, lower dword				0x0684		
Transmit Buffer Address, lower dword				0x0688		
Transmit Buffer Address, upper dword				0x068c		
Transmit Frame Status Word				0x0690		
TCP/IP checksum extension						
Reserved		**Start value**		0x0694		
Start position		**Write position**		0x0698		
Reserved				0x069c		
Current Transmit Descriptor Address, lower dword				0x06a0		13
Current/Next Transmit Descriptor Address, upper dword				0x06a4		
Current Address Counter, lower dword				0x06a8		
Current Address Counter, upper dword				0x06ac		
Current Byte Counter				0x06b0		
BMU Control/Status Register				0x06b4		
Flag Register including FIFO Address Counters				0x06b8		
Test Register 1				0x06bc		
Test Register 2				0x06c0		
Test Register 3				0x06c4		
Reserved				0x06c8 : 0x06fc		

Byte<3>	Byte<2>	Byte<1>	Byte<0>	Memory Address	STI	I/O RAP Block
Sync. Transmit Queue 2						14
Current Transmit Descriptor						
Transmit Buffer Control		**Transmit Buffer Byte Count**		0x0700		
Next Transmit Descriptor Address, lower dword				0x0704		
Transmit Buffer Address, lower dword				0x0708		
Transmit Buffer Address, upper dword				0x070c		
Transmit Frame Status Word				0x0710		
TCP/IP checksum extension						
Reserved		**Start value**		0x0714		
Start position		**Write position**		0x0718		
Reserved				0x071c		
Current Transmit Descriptor Address, lower dword				0x0720		
Current/Next Transmit Descriptor Address, upper dword				0x0724		
Current Address Counter, lower dword				0x0728		
Current Address Counter, upper dword				0x072c		
Current Byte Counter				0x0730		
BMU Control/Status Register				0x0734		
Flag Register including FIFO Address Counters				0x0738		
Test Register 1				0x073c		
Test Register 2				0x0740		
Test Register 3				0x0744		
Reserved				0x0748 : 0x077c		

Byte<3>	Byte<2>	Byte<1>	Byte<0>	Memory Address	STI	I/O RAP Block
Async. Transmit Queue 2						
Current Transmit Descriptor						
Transmit Buffer Control		**Transmit Buffer Byte Count**		0x0780		
Next Transmit Descriptor Address, lower dword				0x0784		
Transmit Buffer Address, lower dword				0x0788		
Transmit Buffer Address, upper dword				0x078c		
Transmit Frame Status Word				0x0790		
TCP/IP checksum extension						
Reserved		**Start value**		0x0794		
Start position		**Write position**		0x0798		
Reserved				0x079c		
Current Transmit Descriptor Address, lower dword				0x07a0		15
Current/Next Transmit Descriptor Address, upper dword				0x07a4		
Current Address Counter, lower dword				0x07a8		
Current Address Counter, upper dword				0x07ac		
Current Byte Counter				0x07b0		
BMU Control/Status Register				0x07b4		
Flag Register including FIFO Address Counters				0x07b8		
Test Register 1				0x07bc		
Test Register 2				0x07c0		
Test Register 3				0x07c4		
Reserved				0x07c8 : 0x07fc		

Byte<3>	Byte<2>	Byte<1>	Byte<0>	Memory Address	STI	I/O RAP Block
Receive RAMbuffer 1						
Receive Rambuffer Start Address				0x0800		
Receive Rambuffer End Address				0x0804		
Receive Buffer Write Pointer				0x0808		
Receive Buffer Read Pointer				0x080c		
Receive Rambuffer Upper Threshold/Pause Packets				0x0810		
Receive Rambuffer Lower Threshold/Pause Packets				0x0814		
Receive Rambuffer Upper Threshold/High Priority				0x0818		
Receive Rambuffer Lower Threshold/High Priority				0x081c		
Receive Rambuffer Packet Counter				0x0820		
Receive Rambuffer Level				0x0824		
Receive Rambuffer Control/Test				0x0828		
Reserved				0x082c : 0x087c		

Byte<3>	Byte<2>	Byte<1>	Byte<0>	Memory Address	STI	I/O RAP Block
Receive RAMbuffer 2						17
Receive Rambuffer Start Address				0x0880		
Receive Rambuffer End Address				0x0884		
Receive Buffer Write Pointer				0x0888		
Receive Buffer Read Pointer				0x088c		
Receive Rambuffer Upper Threshold/Pause Packets				0x0890		
Receive Rambuffer Lower Threshold/Pause Packets				0x0894		
Receive Rambuffer Upper Threshold/High Priority				0x0898		
Receive Rambuffer Lower Threshold/High Priority				0x089c		
Receive Rambuffer Packet Counter				0x08a0		
Receive Rambuffer Level				0x08a4		
Receive Rambuffer Control/Test				0x08a8		
Reserved				0x08ac : 0x08fc		
Reserved	Reserved	Reserved		0x0900 **:** 0x09fc		18 : 19

Byte<3>	Byte<2>	Byte<1>	Byte<0>	Memory Address	STI	I/O RAP Block
Sync. Transmit RAMbuffer 1						
Transmit Rambuffer Start Address				0x0a00		
Transmit Rambuffer End Address				0x0a04		
Transmit Buffer Write Pointer				0x0a08		
Transmit Buffer Read Pointer				0x0a0c		
Reserved				0x0a10		
Reserved				0x0a14		20
Reserved				0x0a18		
Reserved				0x0a1c		
Transmit Rambuffer Packet Counter				0x0a20		
Transmit Rambuffer Level				0x0a24		
Transmit Rambuffer Control/Test				0x0a28		
Reserved				0x0a2c : 0x0a7c		

Byte<3>	Byte<2>	Byte<1>	Byte<0>	Memory Address	STI	I/O RAP Block
Async. Transmit RAMbuffer 1						21
Transmit Rambuffer Start Address				0x0a80		
Transmit Rambuffer End Address				0x0a84		
Transmit Buffer Write Pointer				0x0a88		
Transmit Buffer Read Pointer				0x0a8c		
Reserved				0x0a90		
Reserved				0x0a94		
Reserved				0x0a98		
Reserved				0x0a9c		
Transmit Rambuffer Packet Counter				0x0aa0		
Transmit Rambuffer Level				0x0aa4		
Transmit Rambuffer Control/Test				0x0aa8		
Reserved				0x0aac : 0x0afc		

Byte<3>	Byte<2>	Byte<1>	Byte<0>	Memory Address	STI	I/O RAP Block
Sync. Transmit RAMbuffer 2						
Transmit Rambuffer Start Address				0x0b00		
Transmit Rambuffer End Address				0x0b04		
Transmit Buffer Write Pointer				0x0b08		
Transmit Buffer Read Pointer				0x0b0c		
Reserved				0x0b10		
Reserved				0x0b14		22
Reserved				0x0b18		
Reserved				0x0b1c		
Transmit Rambuffer Packet Counter				0x0b20		
Transmit Rambuffer Level				0x0b24		
Transmit Rambuffer Control/Test				0x0b28		
Reserved				0x0b2c : 0x0b7c		

Byte<3>	Byte<2>	Byte<1>	Byte<0>	Memory Address	STI	I/O RAP Block
Async. Transmit RAMbuffer 2						23
Transmit Rambuffer Start Address				0x0b80		
Transmit Rambuffer End Address				0x0b84		
Transmit Buffer Write Pointer				0x0b88		
Transmit Buffer Read Pointer				0x0b8c		
Reserved				0x0b90		
Reserved				0x0b94		
Reserved				0x0b98		
Reserved				0x0b9c		
Transmit Rambuffer Packet Counter				0x0ba0		
Transmit Rambuffer Level				0x0ba4		
Transmit Rambuffer Control/Test				0x0ba8		
Reserved				0x0bac : 0x0bfc		

Byte<3>	Byte<2>	Byte<1>	Byte<0>	Memory Address	STI	I/O RAP Block
Receive MAC FIFO 1						
Receive MAC FIFO End Address				0x0c00		
Receive MAC FIFO Write Pointer				0x0c04		
Reserved				0x0c08		
Receive MAC FIFO Read Pointer				0x0c0c		
Receive MAC FIFO Packet Counter				0x0c10		
Receive MAC FIFO Level				0x0c14		
Receive MAC Control				0x0c18		
Receive MAC FIFO Control/Test				0x0c1c		
RX LED Counter Init Value				0x0c20		
RX LED Counter				0x0c24		
Reserved	Reserved	**RX LED Counter Test**	**RX LED Counter Control**	0x0c28		
Reserved				0x0c2c		
LINK_SYNC Counter Init Value				0x0c30		
LINK_SYNC Counter				0x0c34		
Reserved	Reserved	**LINK_SYNC Counter Test**	**LINK_SYNC Counter Control**	0x0c38		
Reserved	Reserved	Reserved	**Link LED Reg.**	0x0c3c		
Reserved				0x0c40 : 0x0c7c		

Byte<3>	Byte<2>	Byte<1>	Byte<0>	Memory Address	STI	I/O RAP Block
Receive MAC FIFO 2						
Receive MAC FIFO End Address				0x0c80		
Receive MAC FIFO Write Pointer				0x0c84		
Reserved				0x0c88		
Receive MAC FIFO Read Pointer				0x0c8c		
Receive MAC FIFO Packet Counter				0x0c90		
Receive MAC FIFO Level				0x0c94		
Receive MAC Control				0x0c98		
Receive MAC FIFO Control/Test				0x0c9c		
RX LED Counter Init Value				0x0ca0		
RX LED Counter				0x0ca4		
Reserved	Reserved	**RX LED Counter Test**	**RX LED Counter Control**	0x0ca8		
Reserved				0x0cac		
LINK_SYNC Counter Init Value				0x0cb0		
LINK_SYNC Counter				0x0cb4		
Reserved	Reserved	**LINK_SYNC Counter Test**	**LINK_SYNC Counter Control**	0x0cb8		
Reserved	Reserved	Reserved	**Link LED Reg.**	0x0cbc		
Reserved				0x0c9c0 : 0x0cfc		

Byte<3>	Byte<2>	Byte<1>	Byte<0>	Memory Address	STI	I/O RAP Block
Transmit MAC FIFO 1						
Transmit MAC FIFO End Address				0x0d00		
Transmit MAC FIFO Write Pointer				0x0d04		
Transmit MAC FIFO Write Shadow Pointer				0x0d08		
Transmit MAC FIFO Read Pointer				0x0d0c		
Transmit MAC FIFO Packet Counter				0x0d10		
Transmit MAC FIFO Level				0x0d14		
Transmit MAC Control				0x0d18		
Transmit MAC FIFO Control/Test				0x0d1c		
TX LED Counter Init Value				0x0d20		
TX LED Counter				0x0d24		
Reserved	Reserved	**TX LED Counter Test**	**TX LED Counter Control**	0x0d28		
Reserved				0x0d2c		
Reserved				0x0d30 : 0x0d7c		

Byte<3>	Byte<2>	Byte<1>	Byte<0>	Memory Address	STI	I/O RAP Block
Transmit MAC FIFO 2						
Transmit MAC FIFO End Address				0x0d80		
Transmit MAC FIFO Write Pointer				0x0d84		
Transmit MAC FIFO Write Shadow Pointer				0x0d88		
Transmit MAC FIFO Read Pointer				0x0d8c		
Transmit MAC FIFO Packet Counter				0x0d90		
Transmit MAC FIFO Level				0x0d94		
Transmit MAC Control				0x0d98		
Transmit MAC FIFO Control/Test				0x0d9c		27
TX LED Counter Init Value				0x0da0		
TX LED Counter				0x0da4		
Reserved	Reserved	**TX LED Counter Test**	**TX LED Counter Control**	0x0da8		
Reserved				0x0dac		
Reserved				0x0db0 : 0x0dfc		

Byte<3>	**Byte<2>**	**Byte<1>**	**Byte<0>**	**Memory Address**	**STI**	**I/O RAP Block**
Descriptor Poll Timer						
Reserved	**Timer Init Value**			0x0e00		28
Reserved	**Timer**			0x0e04		
Reserved	**Timer Test**	Reserved	**Timer Control**	0x0e08		
Reserved				0x0e0c		
Reserved				0x0e10 : 0x0e8c		
Reserved	Reserved	Reserved		0x0e90 **:** 0x0ffc		29 : 31
Reserved	Reserved	Reserved		0x1000 **:** 0x1ffc		0x20 : 0x3f
Reserved	Reserved	**XMACII 1 Register <0x00>** **.** **.** **XMACII 1 Register <0x3ff>** addresses 0x00:0x3ff follow XMACII datasheet		0x2000 **:** 0x27fc		0x40 : 0x4f
Reserved	Reserved	Reserved		0x2800 **:** 0x2ffc		0x50 : 0x5f

Byte<3>	**Byte<2>**	**Byte<1>**	**Byte<0>**	**Memory Address**	**STI**	**I/O RAP Block**
Reserved	Reserved	**XMACII 2 Register <0x00>** **.** **.** **XMACII 2 Register <0x3ff>** addresses 0x00:0x3ff follow XMACII datasheet		0x3000 **:** 0x37fc		0x60 : 0x6f
Reserved	Reserved	Reserved		0x3800 **:** 0x3ffc		0x70 0x7f

12.0 Registers

This section lists the registers and their usage down to bit level.

12.1 BLock 0

Block 0 is I/O mapped into the lower half of the 256 bytes I/O space permanently. It may be mapped into the upper half too by setting the Register Address Port (RAP) to 0.

12.1.1 Register Address Port (RAP)

The RAP controls the mapping of the upper 128 bytes in I/O space. The RAP itself is contained in block 0 wich is mapped to the lower 128 bytes in I/O space permanently.

Bit	Name	Description	Write	Read	Reset (SW)
	Register Address Port				
31:7	Reserved				
6:0	**RAP**	Specifies one out of block 0 - 127, which is mapped to the upper half of the 256byte I/O range. 0=block 0,... ,0x7f=block 127	yes	aw	0

12.1.2 LED Register

Bit	Name	Description	Write	Read	Reset (SW)
	LED Registers				
7:2	Reserved				
1	**LED<0>On**	LED (middle, green) On/ Off	exec	0b10	0
0	**LED<0> Off**			0b01	1

12.1.3 Control/Status Register

Bit	Name	Description	Write	Read	Reset
Control/Status Register					
	Status				
15:10	Reserved				
9	**Bus Clock**	**Bus Clock** is the **M66EN** Pin of the PCI Bus used as input 0 = clock speed 33MHz or lower 1 = clock speed between 33MHz and 66MHz	ne	value	
8	**Slot Size**	**Slot Size** is **REQ64**# latched with the rising edge of **RST#** and inverted. 0 = network interface card installed in 32-bit slot 1 = network interface card installed in 64-bit slot	ne	value	
	Commands				
7	**Set IRQ SW**	Sets and clears Interrupt Request from SW	exec	0b10	0
6	**Clear IRQ SW**			0b01	1
5	**Stop Master Done**	As soon as the master statemachine is in the idle state after **Stop Master** is set, **Stop Master Done** is asserted. **Stop Master Done** is reset to 0 by resetting **Stop Master**.	ne	value	0
4	**Stop Master**	If **Stop Master** is set, all requests from the BMUs except for the one being serviced at the moment, are masked. The master state-machine will reach the idle state after the current request is serviced. **Stop Master** has to be reset by the software after the BMUs are reset. If the BMUs aren't reset, the network interface card will resume master action at the point, where it was interrupted by **Stop Master**.	yes	aw	0 (HW)

Bit	Name	Description	Write	Read	Reset
3	**Master Reset Clear**	If **Master Reset** is set, all devices related to the master interface (BMUs, FIFOs, State machines) are in their reset state. Set/Clear **Master Reset.** executed, if appropriate bit is set to 1.	exec	0b10	0
2	**Master Reset Set**			0b01	1 (SW)
1	**SW Reset Clear**	Set/Clear **SW Reset.** executed, if appropriate bit is set to 1. If **SW Reset** is set, all internal and external devices are in their reset state.	exec	0b10	0
0	**SW Reset Set**			0b01	1 (HW)

12.1.4 Interrupt Source Registers

If set to ONE, interrupt is pending.

Bit	Name	Description	Write	Read	Reset (SW)
	Interrupt Source Register				
31	**IRQ HW Error**	Interrupt Hardware Error see Interrupt HW Error Source Register	ne	value	0
30	Reserved		ne	value	0
29	**IRQ Pkt Tout Rx1**	Interrupt Packet Timeout (receive queue 1) Transfer from XMACII is hanging	ne	value	0
28	**IRQ Pkt Tout Rx2**	Interrupt Packet Timeout (receive queue 2) Transfer from XMACII is hanging	ne	value	0
27	**IRQ Pkt Tout Tx1**	Interrupt Packet Timeout (transmit queue 1) Transfer from XMACII is hanging	ne	value	0
26	**IRQ Pkt Tout Tx2**	Interrupt Packet Timeout (transmit queue 2) Transfer from XMACII is hanging	ne	value	0
25	**IRQ I2C Ready**	Interrupt on completion of I^2C transfer	ne	value	0

Bit	Name	Description	Write	Read	Reset (SW)
24	**IRQ SW**	Interrupt set by SW in control register	ne	value	0
23	**IRQ External Reg.**	Interrupt from external register This external interrupt line is connected to the interrupt out put of the voltage/temperature sensor	ne	value	0
22	**IRQ Timer**	Interrupt timer	ne	value	0
21	**IRQ MAC 1**	Interrupt from MAC 1	ne	value	0
20	**IRQ LINK_SYNC Counter MAC 1**	Interrupt LINK_SYNC counter wrapped (MAC 1)	ne	value	0
19	**IRQ MAC 2**	Interrupt from MAC 2	ne	value	0
18	**IRQ LINK_SYNC Counter MAC 2**	Interrupt LINK_SYNC counter wrapped (MAC 2)	ne	value	0
17	**IRQ EOB Rx 1**	Interrupt End Of Buffer (receive queue 1)	ne	value	0
16	**IRQ EOF Rx 1**	Interrupt End Of Frame (receive queue 1)	ne	value	0
15	**IRQ CHCK Rx 1**	Interrupt coding error of descriptor (receive queue 1)	ne	value	0
14	**IRQ EOB Rx 2**	Interrupt End Of Buffer (receive queue 2)	ne	value	0
13	**IRQ EOF Rx 2**	Interrupt End Of Frame (receive queue 2)	ne	value	0
12	**IRQ CHCK Rx 2**	Interrupt coding error of descriptor (receive queue 2)	ne	value	0
11	**IRQ EOB s. TX 1**	Interrupt End Of Buffer (synchronous transmit queue 1)	ne	value	0
10	**IRQ EOF s. TX 1**	Interrupt End Of Frame (synchronous transmit queue 1)	ne	value	0

Bit	Name	Description	Write	Read	Reset (SW)
9	**IRQ CHCK s. TX 1**	Interrupt coding error of descriptor (synchronous transmit queue 1)	ne	value	0
8	**IRQ EOB as. TX 1**	Interrupt End Of Buffer (asynchronous transmit queue 1)	ne	value	0
7	**IRQ EOF as. TX 1**	Interrupt End Of Frame (asynchronous transmit queue 1)	ne	value	0
6	**IRQ CHCK as. TX 1**	Interrupt coding error of descriptor (asynchronous transmit queue 1)	ne	value	0
5	**IRQ EOB s. TX 2**	Interrupt End Of Buffer (synchronous transmit queue 2)	ne	value	0
4	**IRQ EOF s. TX 2**	Interrupt End Of Frame (synchronous transmit queue 2)	ne	value	0
3	**IRQ CHCK s. TX 2**	Interrupt coding error of descriptor (synchronous transmit queue 2)	ne	value	0
2	**IRQ EOB as. TX 2**	Interrupt End Of Buffer (asynchronous transmit queue 2)	ne	value	0
1	**IRQ EOF as. TX 2**	Interrupt End Of Frame (asynchronous transmit queue 2)	ne	value	0
0	**IRQ CHCK as. TX 2**	Interrupt coding error of descriptor (asynchronous transmit queue 2)	ne	value	0

Bit	Name	Description	Write	Read	Reset (SW)
	Interrupt HW Error Source Register				
31:12	Reserved				
11	**IRQ Master Error**	Interrupt master error detected on master accesses Set, if **DATAPERR**, **RTABORT** or **RMABORT** are set.	ne	value	0
10	**IRQ Status**	Interrupt status exception Set, if **PERR, RMABORT, RTABORT** or **DATAPERR** are set.	ne	value	0
9	**IRQ No Status MAC 1**	Interrupt No receive Status from MAC 1 Set, if **Status Valid** is not set (see receive descriptor)	ne	value	0
8	**IRQ No Status MAC 2**	Interrupt No receive Status from MAC 2 Set, if **Status Valid** is not set (see receive descriptor)	ne	value	0
7	**IRQ No Timestamp MAC 1**	Interrupt No receive Status from MAC 1 Set, if **Time Stamp Valid** is not set (see receive descriptor)	ne	value	0
6	**IRQ No Timestamp MAC 2**	Interrupt No receive Status from MAC 2 Set, if **Time Stamp Valid** is not set (see Receive descriptor)	ne	value	0
5	**IRQ PAR Rd Ram**	Interrupt Parity error/RAM Interface, on Read This interrupt is intended to indicate a panic event (hardware fault)	ne	value	0
4	**IRQ PAR Wr Ram**	Interrupt Parity error/RAM Interface, on Write This interrupt is intended to indicate a panic event (hardware fault)	ne	value	0
3	**IRQ Par MAC 1**	Interrupt Parity error/MAC 1 This interrupt is intended to indicate a panic event (hardware fault)	ne	value	0

Bit	Name	Description	Write	Read	Reset (SW)
2	**IRQ Par MAC 2**	Interrupt Parity error/MAC 2 This interrupt is intended to indicate a panic event (hardware fault)	ne	value	0
1	**IRQ PAR Rx 1**	Interrupt Parity error (receive queue 1) This interrupt is intended to indicate a panic event (hardware fault)	ne	value	0
0	**IRQ PAR Rx 2**	Interrupt Parity Error (receive queue 2) This interrupt is intended to indicate a panic event (hardware fault)	ne	value	0

12.1.5 Interrupt Mask Registers

Each bit position defines whether the dedicated interrupt is propagated to the interrupt line **INTA#**.

The enable bits have the same bit positions as in the **Interrupt Source Registers.**

If set to ONE, interrupt is enabled.

See also "**Special Interrupt Source Register ".**

Bit	Name	Description	Write	Read	Reset (SW)
	Interrupt Mask Register				
31:0	**En IRQ xxx**	Enable Interrupt xxx	yes	aw	0

12.1.6 Special Interrupt Source Register

This register is mirroring the **Interrupt Source Register** with special functionality adapted to typical software handling.

If the interrupt line **INTA#** is asserted, the read value is the same as in the **Interrupt Source Register**.
If the interrupt line **INTA#** is NOT asserted, the read value is 0.
If the interrupt line **INTA#** is asserted, reading the **Special Interrupt Source Register** clears the **Interrupt Mask Register** (NOT the **Interrupt HW Error Mask Register**). As a result the interrupt line **INTA#** is deasserted.

Bit positions are the same as in the **Interrupt Source Register.**

12.1.7 Special Interrupt Mask Register

Bit	Name	Description	Write	Read	Reset (SW)
	Special Interrupt Mask Register				
31:0	**IRQ xxx**	Interrupt xxx Read clears the **Interrupt Mask Register** (NOT the **Interrupt HW Error Mask Register**). As a result the interrupt line **INTA#** is deasserted.	ne	value or 0	0

12.1.8 Interrupt Hardware Error Mask Register

Bit	Name	Description	Write	Read	Reset (SW)
	Interrupt Hardware Error Mask Register				
31:0	**En IRQ xxx**	Enable Hardware Interrupt xxx	yes	aw	0

12.1.9 Interrupt Moderation Mask Registers

Each bit position defines, if the dedicated interrupt is moderated or propagated directly to the interrupt line **INTA#**. The enable bits have the same bit positions as in the **Interrupt Source Registers.**

If set to ONE, interrupt is moderated.

If set to ZERO, interrupt is propagated directly to the interrupt line **INTA#.**

Bit	Name	Description	Write	Read	Reset (SW)
	Interrupt Moderation Mask Register				
31:0	**En Mod IRQ xxx**	Enable moderation interrupt xxx	yes	aw	0

Bit	Name	Description	Write	Read	Reset (SW)
	Interrupt Hardware Error Moderation Mask Register				
31:0	**En Mod IRQ xxx**	Enable Moderation Hardware Interrupt xxx	yes	aw	0

12.1.10 IRQ Moderation Timer Registers

(IM is an abbreviation for Interrupt Moderation)

Bit	Name	Description	Write	Read	Reset (SW)
31:0	**IRQ Moderation Timer Init Value**		yes	aw	0
31:0	**IRQ Moderation Timer**		yes	value	0
	IRQ Moderation Timer Control/Test				
31:16	Reserved				
		Test			
15:11	Reserved				
10	**IM Timer Test On**	Testmode ON/OFF	exec	0b10	0
9	**IM Timer Test Off**			0b01	1
8	**IM Timer Step**	Timer decrement	exec	0	
		Control			
7:3	Reserved				
2	**IM Timer Start**	Start/Stop Timer	exec	0b10	0
1	**IM Timer Stop**			0b01	1
0	Reserved				

The IRQ moderation timer implements write posting and retries the following accesses to the interrupt moderation timer while a posted write is in progress.

Target reads are retried until the addressed register is synchronized to PCI Clock.

12.1.11 MAC-Address Registers

These registers are holding the MAC Address.

They are loaded at POWER ON RESET from the Flash EPROM. They may be written in testmode.

Bit	Name	Description	Write	Read	Reset (HW)
MAC-Address Registers					
31:16	Reserved				
15:8	**MAC<5>**	Mac-Address, Byte 5	ne	value	0
7:0	**MAC<4>**	Mac-Address, Byte 4	ne	value	0
31:24	**MAC<3>**	Mac-Address, Byte 3	ne	value	0
23:16	**MAC<2>**	Mac-Address, Byte 2	ne	value	0
15:8	**MAC<1>**	Mac-Address, Byte 1	ne	value	0
7:0	**MAC<0>**	Mac-Address, Byte 0	ne	value	0

12.1.12 Interface Type Register

This register is holding the type of interface.

It is loaded at POWER ON RESET from the Flash EPROM. They may be written in testmode.

Bit	Name	Description	Write	Read	Reset (HW)
Interface Type Register					
31:24	**Chip Revision**	Initial Revision is 0x0a Fixed Value	ne	0x0a	
	Configuration				
23:18	Reserved				

Bit	Name	Description	Write	Read	Reset (HW)
17	**Dis DL Clocks**	Disable clocks for 2nd MAC 1= disable 0= enable (Default MUST be “enable” guaranteeing clocks at POWER ON RESET)	ne	value	0
16	**Single MAC**	MAC configuration flag 1= no 2nd MAC available 0= 2nd MAC available Single MAC is for software purposes only	ne	value	0
15:8	**PMD**	PMD Type	ne	value	0
7:0	**Connector**	Connector Type	ne	value	0

The following values for PMD type encoding are defined by SysKonnect. Not all types PMDs are available for the current version of the **“SK-NET GENESIS <x>”** network interface card)

.

PMD type	Code letter	ASCII	Description
1000Base-LX	L	0x4c	long wavelength laser
1000Base-SX	S	0x53	short wavelength laser
1000Base-CX	C	0x43	short copper jumpers
1000Base-T	T	0x54	4-pair UTP Cat.5
undefined (SK proprietary)		0x00	not specified

:The various connector types are encoded as shown in the following table. (Not all these connector types are in use with the **"SK-NET GENESIS <x>"** network interface card):

Connector type	Code letter	ASCII	Short Name
Duplex SC type MIC	C	0x43	Duplex-SC
Shielded D-Sub MIC	D	0x44	D-Sub
Shielded ANSI Fiber Channel connector style 2	F	0x46	FC S2
UTP MIC	J	0x4a	UTP
undefined / all others (SK proprietary)		0x00	unknown

12.1.13 Flash EPROM Registers

These registers are holding optional information. They are loaded at POWER ON RESET from the Flash EPROM. They may be written in testmode

Bit	Name	Description	Write	Read	Reset (HW)
EPROM Registers					
31:24	**Eprom<3>**	EPROM, Byte 3	ne	value	0
23:16	**Eprom<2>**	EPROM, Byte 2	ne	value	0
15:8	**Eprom<1>**	EPROM, Byte 1	ne	value	0
7:0	**Eprom<0>**	EPROM, Byte 0	ne	value	0

Flash EPROM Register <0>

EPROM <0> is used for the encoding of the mounted external buffer memory. The following values are assigned:

Value	External Memory	Offset[1] for RAMbuffer Addressing	Mount Options
EPROM Register <0>			
0x01	512 KBytes	0x0	64KBytes SRAMs

Value	External Memory	Offset[1] for RAMbuffer Addressing	Mount Options
0x02	1024 KBytes	0x0	128KBytes SRAMs
0x03	1024 KBytes	0x80000	64KBytes SRAMs
0x04	2048 KBytes	0x0	128KBytes SRAMs

1. Offset is counted in multiples of bytes

12.1.14 EPROM Address Register/Counter

Bit	Name	Description	Write	Read	Reset (HW)
EPROM Address Register/Counter					
31:17	Reserved				
16:0	**Address Register/Counter**	Defines 17-bit Flash EPROM address	yes	aw	0x1ffff
EPROM Data Register					
31:24	Reserved				
23:16	Reserved				
15:8	Reserved				
7:0	**Data Port**	Programming EPROM data port	exec	value	0
EPROM Loader Control Register					
31:16	Reserved				
		Test			
15:12	Reserved				
11	**Loader Test On**	Testmode On/Off	exec	0b10	0
10	**Loader Test Off**			0b01	1
9	**Loader Step**	Decrement **EPROM Address Counter**.	exec	0	0
8	**Loader Start**	Starts loading of Flash EPROM at the location defined by the **Address Register**.	exec	0	0

Bit	Name	Description	Write	Read	Reset (HW)
		Control			
7:0	Reserved				

12.1.15 Timer Registers

Bit	Name	Description	Write	Read	Reset (SW)
31:0	**Timer Init Value**		yes	aw	0
31:0	**Timer**		yes (for tests only)	value	0
	Timer Control/Test				
31:16	Reserved				
		Test			
15:11	Reserved				
10	**Timer Test On**	Testmode ON/OFF	exec	0b10	0
9	**Timer Test Off**			0b01	1
8	**Timer Step**	Timer decrement	exec	0	
		Control			
7:3	Reserved				
2	**Timer Start**	Start/Stop timer	exec	0b10	0
1	**Timer Stop**			0b01	1
0	**Timer Clear IRQ**	Clear timer interrupt	exec	0	

The timer implements write posting and retries the following accesses to the timer while a posted write is in progress. Target reads are retried until the addressed register is synchronized to PCI Clock.

12.1.16 Test Control Registers

Bit	Name	Description	Write	Read	Reset (SW)
Test Control Register 1 / Commands					
7	**Force DataPERR Master Read**	Simulate data parity error on **PAR** on next master read access	exec	0	0
6	**Force DataPERR Master Write**	Generate data parity error on **PAR** on next master write access	exec	0	0
5	**Force DataPERR Target Read**	Generate data parity error on **PAR** on next target read access	exec	0	0
4	**Force DataPERR Target Write**	Simulate data parity error on **PAR** on **this** target write access	exec	0	0
3	**Force AddrPERR Master**	Generate address parity error on **PAR** on first address phase of next master access	exec	0	0
2	**Force AddrPERR Target**	Simulate address parity error on next target access	exec	0	0
1	**En Config Write On**	Enables write accesses to the **Configuration Registers** over the **Control Register File**.	exec	0b10	0
0	**En Config Write Off**			0b01	1

Bit	Name	Description	Write	Read	Reset (SW)
Test Control Register 2 / Commands					
7:4	Reserved				
3	**Force DataPERR Master Read 64**	Simulate data parity error on **PAR64** on next master read access	exec	0	0

Bit	Name	Description	Write	Read	Reset (SW)
2	**Force DataPERR Master Write 64**	Generate data parity error on **PAR64** on next master write access	exec	0	0
1	**Force AddrPERR Master 64**	Generate address parity error on **PAR64** on first address phase of next master Dual Address Cycle	exec	0	0
0	**Force AddrPERR Master 2nd**	Generate address parity error on **PAR** on second address phase of next master Dual Address Cycle	exec	0	0

The commands for forcing parity errors are executed once on the same or next access. Status bits are set and interrupts are generated as defined by PCI and/or the related description in this document.
Any output parity errors generated by the network interface card (master address, master write data, target read data) can cause reactions from the target (SERR#, PERR#), which may cause the system to halt. diagnostic software has to keep this in mind.
Any input parity errors detected by the network interface card (target address, master read data, target write data) will cause the assertion of PERR# or SERR#, if enabled, and the setting of the appropriate bits in the Config Status Register by the network interface card. To avoid all unnecessary system hang, the assertion of PERR# and SERR# by the network interface card is suppressed, when executing one of these commands. To maintain system integrity the diagnostic software must reset the corresponding config status bits after each simulated parity error.

Handle with care.

12.1.17 General Purpose I/O Registers

For further extension the General Purpose I/O Pins are routed to the General Purpose Registers. These I/Os are programmable as inputs or outputs

Bit	Name	Description	Write	Read	Reset value
31:26	Reserved				

Bit	Name	Description	Write	Read	Reset value
25:16	**GPIO Dir<9:0>**	Defines the type of the General Purpose I/O Pins. 1 = output 0 = input	yes	aw	0
15:10	Reserved				
9:0	**GPIO<9:0>**	These bits are routed to the ASIC's pins for future external options. As output they are synchronous to PCI CLK, As input they are synchronized to PCI CLK.	yes	aw	0

12.1.18 I^2C (HW) Registers

Bit	Name	Description	Write	Read	Reset value
	I^2C (HW) Control Register				
31	**Flag**	Starts the I^2C data transfers, determines its direction and signals its completion by being toggled by HW. If written 1, a I^2C write is started, will be set to 0 after completion. If written 0, a I^2C read is started, will be set to 1 after completion. Generates an interrupt on completion.	exec	value	0
30:16	**I2C Address**	Address of the I^2C device register to be written / read.	yes	aw	0x00
15:9	**I2C Devsel**	Devsel Byte of the I^2C device to be written / read.	yes	aw	0x00
8:5	Reserved				
4	**I2C Burst**	0 = single Byte transfers 1 = 4 Byte Page Mode write transfers with fixed page size of 8 Bytes assumed	yes	aw	0

Bit	Name	Description	Write	Read	Reset value
3:1	**I2C Device Size**	Defines the size of the addressed I^2C Device in Bytes. 0 = 256 Bytes and smaller 1 = 512 Bytes 2 = 1024 Bytes 3 = 2048 Bytes 4 = 4096 Bytes 5 = 8192 Bytes 6 = 16384 Bytes 7 = 32768 Bytes Default value is 256 Bytes.	yes	aw	0x00
0	**I2C Stop**	If written 1, interrupts the current I^2C transfer at the next byte boundary with a stop condition and signals end of I^2C transfer by toggling **Flag**.	exec	0	0
	I2C (HW) Data Register				
31:0	**I2C Data**	Must be written before I^2C Address Register for I^2C write. Contains I^2C read Data after completion of I^2C read.	yes	aw	0x00
	I2C (HW) IRQ Register				
31:1	Reserved				
0	**Clear IRQ I2C**	Clears Interrupt Request from I^2C HW interface	exec	0	0

This registers are implementing a serial standard I^2C interface to the optional temperature/voltage sensor. HW runs the 100kHz serial I^2C protocol to obtain data.

The HW controlled I^2C interface and the SW controlled I^2C interface are connected to the same I^2C bus (Pins VPD_DATA and VPD_Clock). They **MUST NOT** be used in parallel.

If the SW controlled I^2C interface is used, the **I2C (SW) Register** has to be set to inactive values (Reset values).

If the HW controlled I^2C interface is used, the SW controlled I^2C interface MUST NOT be started (**FLAG/I2C (SW) Register).**

The I^2C clock and data port pins are pulled high by a pull up resistor to VCC of the I^2C device.

12.1.19 I²C (SW) Register

Bit	Name	Description	Write	Read	Reset (SW)
I²C (SW) Register					
31:3	Reserved				
2	**I2C Data Dir**	Defines direction of I²C Data Port: 0 = Input 1 = Output set to	yes	aw	0
1	**I2C Data**	I²C Interface Data Port	yes	dir=0: value dir=1: aw	0
0	**I2C Clock**	I²C Interface clock	yes	aw	1

This register implements a serial standard I²C interface to the optional temperature/voltage sensor. Software has to run the serial I²C protocol to obtain data.

As output, the **Data Port** must simulate an open collector output in order to obtain a 0.7Vcc signal level at the I²C device (if supplied with 5V).

Driving to low level: **I2C Data** = 0
I2C Data Dir = 1

Floating to high level: **I2C Data** = x
I2C Data Dir = 0

The hardware controlled I²C interface and the software controlled I²C interface are connected to the same I²C bus (Pins I2C DATA and I2C Clock). They **MUST NOT** be used in parallel.

If the software controlled I²C interface is used, the **I2C (SW) Register** has to be set to inactive values (Reset values).

If the Hardware controlled I²C interface is used, the software controlled I²C interface MUST NOT be started (**FLAG/I2C (SW) Register).**

The I²C clock and data port pins are pulled high by a pull up resistor to VCC of the I²C device.

Note: The VPD-Interface come's with another dedicated I²C interface.

12.1.20 RAM Random Registers

The RAM random register is provided for test purposes. Test-, diagnosis software may use the RAM random register to access data in the external memory.
Depending on the mounted SRAMs an offset different from 0x00 must be added to the RAM address. See section *Flash EPROM Register <0>* on page 152.

Bit	Name	Description	Write	Read	Reset (SW)
	RAM Address				
31:19	Reserved				
18:0	**Ram Address**	Defines RAM address in 64-bit words.	yes	aw	0
31:0	**Data Port/lower dword**	Dataport/lower 32-bit word for exchange of read/write data. On a PCI read access, reading from RAM to the data ports is initiated. The initiating PCI access and subsequent accesses are retried on PCI while reading from RAM	yes	value exec	0
31:0	**Data Port/upper dword**	Dataport/upper 32-bit word for exchange of read/write data. On a PCI write access, writing to RAM from the data ports is initiated. Subsequent PCI accesses are retried on PCI while writing to RAM	yes exec	value	0

12.1.21 RAM Interface Registers

The timeout values are limiting the burst length of data transfers for each requester individually.

Default values must be used.

Bit	Name	Description	Write	Read	Reset (private)
	Timeout init Values 0 - 3				
31:24	**Timeout Init Value 3**	Read state machine receive queue1	yes	aw	32
23:16	**Timeout Init Value 2**	Write state machine synchronous transmit queue 1	yes	aw	32
15:8	**Timeout Init Value 1**	Write state machine asynchronous transmit queue1	yes	aw	32
7:0	**Timeout Init Value 0**	Write state machine receive queue1	yes	aw	32
	Timeout init Values 4 - 7				
31:24	**Timeout Init Value 7**	Write state machine asynchronous transmit queue 2	yes	aw	32
23:16	**Timeout Init Value 6**	Write state machine receive queue2	yes	aw	32
15:8	**Timeout Init Value 5**	Read state machine synchronous transmit queues1	yes	aw	32
7:0	**Timeout Init Value 4**	Read state machine asynchronous transmit queue 1	yes	aw	32
	Timeout init Values 8 - 11				
31:24	**Timeout Init Value 11**	Read state machine synchronous transmit queue 2	yes	aw	32
23:16	**Timeout Init Value 10**	Read state machine asynchronous transmit queue 2	yes	aw	32
15:8	**Timeout Init Value 9**	Read state machine receive queue 2	yes	aw	32
7:0	**Timeout Init Value 8**	Write state machine synchronous transmit queue 2	yes	aw	32
	Timeout Timer				
31:8	Reserved				
7:0	**Timeout Timer**		yes (for tests only)	value	0
	RAM Interface Control/Test				

Bit	Name	Description	Write	Read	Reset (private)
		Test			
31:20	Reserved				
19	**Timeout**	= 1, if Timeout timer = 0	ne	value	1
18	**Timeout Timer Test On**	Testmode ON/OFF	exec	0b10	0
17	**Timeout Timer Test Off**			0b01	1
16	**Timeout Timer Step**	Timer decrement	exec	0	
		Control			
15:10	Reserved				
9	**Clear IRQ PAR Rd Ram**	Clear Parity error on read interrupt	exec	0	
8	**Clear IRQ PAR Wr Ram**	Clear Parity error on write interrupt	exec	0	
7:2	Reserved				
1	**Reset Clear**	Set/Clear **Reset.** executed, if appropriate bit is set to 1.	exec	0b10	0
0	**Reset Set**	If **Reset** is set, all RAM interface functions and registers are in their reset state. **Reset** is forwarded to the RAM Random Access Device in order to avoid a hangup on a target access while the RAM interface is resetted.		0b01	1 (SW)

12.1.22 Blink Source Registers

Bit	Name	Description	Write	Read	Reset (SW)
31:0	**Blink Source Counter Init Value**		yes	aw	0x1dcd650 = 500 ms
31:0	**Blink Source Counter**		yes (for tests only)	value	0
	Blink Source Counter Control/Status//Test				
		Test			
31:19	Reserved				
18	**Blink Source Counter Test On**	Testmode ON/OFF	exec	0b10	0
17	**Blink Source Counter Test Off**			0b01	1
16	**Blink Source Counter Step**	Blink Source Counter decrement	exec	0	
		Status			
15:9	Reserved				
8	**Blink Source**		ne	value	0
		Control			
7:2	Reserved				
1	**Blink Source Counter Start**	Start/Stop Blink Source Counter	exec	0b10	0
0	**Blink Source Counter Stop**			0b01	1

The Blink Source implements write posting and retries the following accesses to the Blink Source while a posted write is in progress.

Target reads are retried until the addressed register is synchronized to PCI Clock.

12.1.23 Link LED Register

Bit	Name	Description	Write	Read	Reset (SW)
Link LED Register					
7:6	Reserved				
5	**Blinking On**	**Controls the mode of the Link LED** **(Truthtable see “LEDs”)**	**exec**	**0b10**	**0**
4	**Blinking Off**			**0b01**	**1**
3	**LINK_SYNC On**		**exec**	**0b10**	**0**
2	**LINK_SYNC Off**			**0b01**	**1**
1	**LED On**		**exec**	**0b10**	**0**
0	**LED Off**			**0b01**	**1**

12.1.24 Transmit Arbiter Registers

Bit	Name	Description	Write	Read	Reset (SW)
31:24	Reserved				
23:0	**Interval Timer Init Value**	Number of clock cycles as time interval for rate control.	yes	aw	0
	Interval Timer				
31:24	Reserved				
23:0	**Interval Timer**		yes	aw	0
	Limit Counter Init Value				
31:24	Reserved				

Bit	Name	Description	Write	Read	Reset (SW)
23:0	**Limit Counter Init Value**	Number of transfer cycles to MAC FIFO in time interval	yes	aw	0
	Limit Counter				
31:24	Reserved				
23:0	**Limit Counter**		yes	aw	0
	Timer/Counter Control/Status/Test				
		Status			
31:17	Reserved				
16	**Priority/Sync. Ram-buffer**	Set, as long as limit counter is not ZERO or if **Force Sync On** is set	ne	value	0
		Test			
15:14	Reserved				
13	**Interval Timer Test On**	Testmode ON/OFF	exec	0b10	0
12	**Interval Timer Test Off**			0b01	1
11	**Interval Timer Step**	Timer decrement	exec	0	
10	**Limit Counter Test On**	Testmode ON/OFF	exec	0b10	0
9	**Limit Counter Test Off**			0b01	1
8	**Limit Counter Step**	Counter decrement	exec	0	
		Control			
7	**Force Sync On**	If On, the Sync RAMbuffer gets highest priority ignoring the timer and counter	exec	0b10	0
6	**Force Sync Off**			0b01	1
5	**En Alloc**	Enable/disable allocation of free bandwidth	exec	0b10	0
4	**Dis Alloc**			0b01	1

Bit	Name	Description	Write	Read	Reset (SW)
3	**Rate Control Start**	Starts/stops rate control (running timer and counter)	exec	0b10	0
2	**Rate Control Stop**			0b01	1
1	**Arbiter Operational On**	Operational mode ON/OFF If OFF, no request is granted. Has to be set to OFF until all other devices are initialized	exec	0b10	0
0	**Arbiter Operational Off**			0b01	1

12.1.25 Packet Arbiter Registers

Bit	Name	Description	Write	Read	Reset (private)
Timeout Values					
	Timeout Init Value Rx1				
31:16	Reserved				
15:0	**Timeout Init Value**		yes	aw	0
	Timeout Init Value Rx2				
31:16	Reserved				
15:0	**Timeout Init Value**		yes	aw	0
	Timeout Init Value Tx1				
31:16	Reserved				
15:0	**Timeout Init Value**		yes	aw	0
	Timeout Init Value Tx2				
31:16	Reserved				
15:0	**Timeout Init Value**		yes	aw	0
Timeout Timers					
	Timeout Timer Rx1				
31:16	Reserved				

Bit	Name	Description	Write	Read	Reset (private)
15:0	**Timeout Timer**		yes (for tests only)	value	0
	Timeout Timer Rx2				
31:16	Reserved				
15:0	**Timeout Timer**		yes (for tests only)	value	0
	Timeout Timer Tx1				
31:16	Reserved				
15:0	**Timeout Timer**		yes (for tests only)	value	0
	Timeout Timer Tx2				
31:16	Reserved				
15:0	**Timeout Timer**		yes (for tests only)	value	0
	Packet Arbiter Control/Test				
		Test			
31	Reserved		ne	value	1
30	**Timeout Timer Tx2 Test On**	Testmode ON/OFF	exec	0b10	0
29	**Timeout Timer Tx2 Test Off**			0b01	1
28	**Timeout Timer Tx2 Step**	Timer decrement	exec	0	
27	Reserved		ne	value	1

Bit	Name	Description	Write	Read	Reset (private)
26	**Timeout Timer Tx1 Test On**	Testmode ON/OFF	exec	0b10	0
25	**Timeout Timer Tx1 Test Off**			0b01	1
24	**Timeout Timer Tx1 Step**	Timer decrement	exec	0	
23	Reserved		ne	value	1
22	**Timeout Timer Rx2 Test On**	Testmode ON/OFF	exec	0b10	0
21	**Timeout Timer Rx2 Test Off**			0b01	1
20	**Timeout Timer Rx2 Step**	Timer decrement	exec	0	
19	Reserved		ne	value	1
18	**Timeout Timer Rx1 Test On**	Testmode ON/OFF	exec	0b10	0
17	**Timeout Timer Rx1 Test Off**			0b01	1
16	**Timeout Timer Rx1 Step**	Timer decrement	exec	0	
		Control			
15:14	Reserved				
13	**Clr IRQ Pkt Tout Tx2**	Clear Interrupt Packet Timeout transmit queue 2	exec	0	0
12	**Clr IRQ Pkt Tout Tx1**	Clear Interrupt Packet Timeout transmit queue 1	exec	0	0
11	**Clr IRQ Pkt Tout Rx2**	Clear Interrupt Packet Timeout receive queue 2	exec	0	0
10	**Clr IRQ Pkt Tout Rx1**	Clear Interrupt Packet Timeout receive queue 1	exec	0	0

Bit	Name	Description	Write	Read	Reset (private)
9	**En Timeout Tx2 On**	Enable Timeout Timer transmit queue 2 executed, if appropriate bit is set to 1.	exec	0b10	0
8	**En Timeout Tx2 Off**			0b01	1
7	**En Timeout Tx1 On**	Enable Timeout Timer transmit queue 1 executed, if appropriate bit is set to 1.	exec	0b10	0
6	**En Timeout Tx1 Off**			0b01	1
5	**En Timeout Rx2 On**	Enable Timeout Timer receive queue 2 executed, if appropriate bit is set to 1.	exec	0b10	0
4	**En Timeout Rx2 Off**			0b01	1
3	**En Timeout Rx1 On**	Enable Timeout Timer receive queue 1 executed, if appropriate bit is set to 1	exec	0b10	0
2	**En Timeout Rx1 Off**			0b01	1
1	**Reset Clear**	Set/Clear **Reset.** executed, if appropriate bit is set to 1. If **Reset** is set, all MAC Arbiter functions and registers are in their reset state.	exec	0b10	0
0	**Reset Set**			0b01	1 (SW)

12.1.26 LINK_SYNC Registers

Bit	Name	Description	Write	Read	Reset (SW)
31:0	**LINK_SYNC Counter Init Value**		yes	aw	0
31:0	**LINK_SYNC Counter**		yes (for tests only)	value	0
	LINK_SYNC Counter/Test				

Bit	Name	Description	Write	Read	Reset (SW)
31:16	Reserved				
		Test			
15:11	Reserved				
10	**LINK_SYNC Counter Test On**	Testmode ON/OFF	exec	0b10	0
9	**LINK_SYNC Counter Off**			0b01	1
8	**LINK_SYNC Counter Step**	Counter decrement	exec	0	
		Control			
7:3	Reserved				
2	**LINK_SYNC Counter Start**	Start/Stop Counter	exec	0b10	0
1	**LINK_SYNC Counter Stop**			0b01	1
0	**LINK_SYNC Counter Clear IRQ**	Clear Counter Interrupt	exec	0	

12.1.27 Rx LED Registers

Bit	Name	Description	Write	Read	Reset (SW)
31:0	**Rx LED Counter Init Value**		yes	aw	0x2faf08 = 50ms
31:0	**Rx LED Counter**		yes (for tests only)	value	0
	Rx LED Counter/Test				
31:16	Reserved				

Bit	Name	Description	Write	Read	Reset (SW)
		Test			
15:11	Reserved				
10	**Rx LED Counter Test On**	Testmode ON/OFF	exec	0b10	0
9	**Rx LED Counter Off**			0b01	1
8	**Rx LED Counter Step**	Counter decrement	exec	0	
		Control			
7:3	Reserved				
2	**Rx LED Counter Start**	Start/Stop Counter	exec	0b10	0
1	**Rx LED Counter Stop**			0b01	1
0	**Rx LED**	=1, if switched on	ne	value	0

12.1.28 Tx LED Registers

Bit	Name	Description	Write	Read	Reset (SW)
31:0	**Tx LED Counter Init Value**		yes	aw	0x2faf08 = 50ms
31:0	**Tx LED Counter**		yes (for tests only)	value	0
	Tx LED Counter/Test				
31:16	Reserved				
		Test			
15:11	Reserved				

Bit	Name	Description	Write	Read	Reset (SW)
10	**Tx LED Counter Test On**	Testmode ON/OFF	exec	0b10	0
9	**Tx LED Counter Off**			0b01	1
8	**Tx LED Counter Step**	Counter decrement	exec	0	
		Control			
7:3	Reserved				
2	**Tx LED Counter Start**	Start/Stop Counter	exec	0b10	0
1	**Tx LED Counter Stop**			0b01	1
0	**Tx LED**	=1, if switched on	ne	value	0

12.1.29 External Registers

For use in other applications where a second external device has to be accessed, an external **Chip Select*/Ready*/IRQ*** interface is implemented. This device may be an 8-bit or 16-bit device. 32 16-bit Register locations are mapped in a subsequent range on 32-bit word boundaries.

The External Registers interface implements write posting. Target reads are retried until the addressed register is synchronized to PCI Clock.

For resetting the external device it may be connected to one of the **XMACII Reset*'s**.

Read data has to be valid before **Ready*** is asserted, write data has to be sampled before **Ready*** is asserted.

The external registers can be accessed with any width at 32-bit word boundaries. On the PCI side all accesses to the external device are completed normally. Accesses without any byte enables asserted or only to the upper 16 bit (neither CBE<1># nor CBE<0># asserted) are treated like accesses to reserved locations (not forwarded to the External Registers).
Note that no byte enables are forwarded to the external registers. Therefore for all write cycles to the external registers all 16 bits are written.

Bit	Name	Description	Write	Read	Reset (SW)
	External Registers		ne, if not available	0xffff, if not available	
Example External Register					
31:16	Reserved				
15:0	**External Register <x>**	External Register 16-bit location<x>	External device's datasheet		
			ne, if not available	0xffff, if not available	

12.1.30 PCI Configuration Registers

The configuration register file is mapped additionally into the control register file.

It is read only with some exceptions: **Our Register 1** and **2** and **VPD Address** and **VPD Data Register** may be written.

Write operations are completed normally on the bus and the data discarded.

For testing purposes, the configuration register file may be set writable by **En Config Write**.

12.1.31 Descriptor Poll Timer Registers

The Poll Timer generates a periodical trigger signal for all BMUs setting **START xxx** such initiating a descriptor read.
This may be enabled for each BMU individually through **En Polling** in the BMU's **Control/Status Registers**.

Bit	Name	Description	Write	Read	Reset (SW)
	Descriptor Poll Timer Init Value				
31:24	Reserved				
23:0	**Init Value**		yes	aw	0
	Descriptor Poll Timer				
31:24	Reserved				
23:0	**Timer**	Multiples of 18.825 ns Max = 315.83 ms	yes (for tests only)	value	0
	Timer Control/Test				
		Test			
31:19	Reserved				
18	**Timer Test On**	Testmode ON/OFF	exec	0b10	0
17	**Timer Test Off**			0b01	1
16	**Timer Step**	Timer decrement	exec	0	
15:8	Reserved				
		Control			
7:2	Reserved				
1	**Timer Start**	Start/Stop Timer	exec	0b10	0
0	**Timer Stop**			0b01	1

The timer implements write posting and retries the following accesses to the timer while a posted write is in progress.

Target reads are retried until the addressed register is synchronized to PCI Clock.

12.1.32 BMU Registers

Each BMU is represented with a nearly identical set of registers.

These registers are intended to be used for testing and diagnostic purposes, except the **Control/Status Registers** and **Next/Current Receive Descriptor Address** and **Watermark** for initialization.

Manipulating the content of these registers under ‘normal’ running conditions is not recommended and may lead to undefined results.

If the default values are acceptable, the **Watermark** should not be written (for future backward compatibility, if watermarks/FIFO depth are adapted).

The **Watermark** of the transmit queues **MUST NOT BE SET** to zero.

Bit	Name	Description	Write	Read	Reset (SW)
Receive Queue Registers					
31:0	**Current Receive Descriptor**		yes for tests only	value	
31:0					
31:0					
31:0					
31:0					
31:0					
31:0					
31:0					
Current Addresses and Counters					
31:0	**Current Receive Descriptor Address, lower dword**		yes	value	0
31:0	**Next/Current Receive Descriptor Address, upper dword** Upper 32 bits for all receive descriptor addresses in 64-bit addressing mode Should be set once at initialization time		yes	value	0
31:0	**Current Address Counter, lower dword** Incrementing address for bus accesses		yes for tests only	value	0
31:0	**Current Address Counter, upper dword** Incrementing address for bus accesses			value	0
	Current Byte Counter				
31:16	Reserved				
15:0	**Byte Counter**	Decrementing byte count for bus accesses		value	0
Receive Queue Control/Status Register					

Bit	Name	Description	Write	Read	Reset (SW)
	Commands				
31:25	Reserved				
24	**Supervisor Idle**	Supervisor SM is in Idle state (see STOP)	ne	value	1
23:22	Reserved				
21	**Reset Desc. Clear**	Set/Clear Reset for Descriptors	exec	0b10	0
20	**Reset Desc. Set**			0b01	1
19	**Reset FIFO Clear**	Set/Clear Reset for FIFOs	exec	0b10	0
18	**Reset FIFO Set**			0b01	1
17	**Run PFI SM**	Reset SM to Idle/Release SM	exec	0b10	0
16	**Reset PFI SM**			0b01	1
15	**Run Supervisor SM**	Reset SM to Idle/Release SM	exec	0b10	0
14	**Reset Supervisor SM**			0b01	1
13	**Run Desc. Read SM**	Reset SM to Idle/Release SM	exec	0b10	0
12	**Reset Desc. Read SM**			0b01	1
11	**Run Desc. Write SM**	Reset SM to Idle/Release SM	exec	0b10	0
10	**Reset Desc. Write SM**			0b01	1
9	**Run Transfer SM**	Reset SM to Idle/Release AM	exec	0b10	0
8	**Reset Transfer SM**			0b01	1
7	**En Polling On**	Enable/Disable polling of descriptors. If enabled, the periodically generated trigger pulse of the **Descriptor Poll Timer** starts the transfer of this Receive Queue (same as **Start Rx**). **NOTE:** After initialization or a successfully completed command **Stop Rx,** polling is started with the next command **Start Rx**.	exec	0b10	0
6	**En Polling Off**			0b01	1

Bit	Name	Description	Write	Read	Reset (SW)
5	**Stop Rx**	Stop Transfer of Receive Queue Stops further reading of descriptors and transferring of data from the Receive Queue (clears internal flag Start RX and forces the Supervisor SM to Idle State). An already running transfer of a receive frame is completed normally until EOF (including writing back the descriptor(s)), even if more than one descriptor/buffer are consumed. The read value is = 1 until stopping is completed. **NOTE**: If there are not enough descriptors available for transferring until EOF, **Stop Rx** becomes not de-asserted. This may be recognized be having **Stop Rx** set while the BMU is running out of descriptors before giving back EOF or having both **Stop Rx** and **Supervisor Idle** set. The only way to get out of this, is to re-issue further descriptors.	exec	0/1	0
4	**Start Rx**	Start Transfer of Receive Queue	exec	0	0
3	**Clr IRQ PAR**	Clear Interrupt Parity	exec	0	0
2	**Clr IRQ EOB**	Clear Interrupt **EOB**	exec	0	0
1	**Clr IRQ EOF**	Clear Interrupt **EOF**	exec	0	0
0	**Clr IRQ ERR**	Clear Interrupt **ERR**	exec	0	0
Receive Queue Flag Register					
31:28	Reserved				
27	**Almost full**	FIFO almost full	ne	value	
26	**EOF in FIFO**	Tag bit in FIFO	ne	value	
25	**Watermark Reached**	Watermark reached in FIFO	ne	value	
24	Reserved				
23:16	**FIFO Level**	Number of 64-bit words used in FIFO	ne	value	
Receive Queue Watermark Register					

Bit	Name	Description	Write	Read	Reset (SW)
15:9	Reserved				
10:0	Watermark	Level for requesting data transfer in bytes	yes	aw	0x100
Receive Queue Test Register 1					
	Supervisor SM				
31:28	**Q<3:0>**	States	if **Load**	value	0
27	**Load**	Load States **Q<3:0>** if = 1, **Q<3:0>** are loaded to SM		0	0
26	**Test On**	Testmode ON/OFF	exec	0b10	0
25	**Test Off**			0b01	1
24	**SM Step**	Step SM	exec	0	
	Read Descriptor SM				
23:20	**Q<3:0>**	States	if **Load**	value	0
19	**Load**	Load States **Q<3:0>** if = 1, **Q<3:0>** are loaded to SM		0	0
18	**Test On**	Testmode ON/OFF	exec	0b10	0
17	**Test Off**			0b01	1
16	**SM Step**	Step SM	exec	0	
	Write Descriptor SM				
15:12	**Q<3:0>**	States	if **Load**	value	0
11	**Load**	Load States **Q<3:0>** if = 1, **Q<3:0>** are loaded to SM		0	0
10	**Test On**	Testmode ON/OFF	exec	0b10	0
9	**Test Off**			0b01	1
8	**SM Step**	Step SM	exec	0	
	Transfer SM				
7:4	**Q<3:0>**	States	if **Load**	value	0
3	**Load**	Load States **Q<3:0>** if = 1, **Q<3:0>** are loaded to SM		0	0
2	**Test On**	Testmode ON/OFF	exec	0b10	0
1	**Test Off**			0b01	1

Bit	Name	Description	Write	Read	Reset (SW)
0	**SM Step**	Step SM	exec	0	
Receive Queue Test Register 2					
31:8	Reserved				
7	**Add. Counter Test On**	Testmode On/Off	exec	0b10	0
6	**Add. Counter Test Off**			0b01	1
5	**Byte Counter Test On**	Testmode On/Off	exec	0b10	0
4	**Byte Counter Test Off**			0b01	1
3:0	**Counter Step**	Decrement **Current Byte Counter** and **Current Address Counter** by value represented by **Counter Step<3:0>**.	exec	0	0
Receive Queue Test Register 3					
31:8	Reserved				
7	Reserved				
6:4	**Mux<2:0>**	Mux position	no	value	0
3	Reserved				
2:0	**VRam<2:0>**	Virtual RAM buffer address	yes	value	0

Bit	Name	Description	Write	Read	Reset (SW)
Transmit Queue Registers					
31:0	**Current Transmit Descriptor**		yes for tests only	value	0
31:0					
31:0					
31:0					
31:0					
31:0					
31:0					
Current Addresses and Counters					

Bit	Name	Description	Write	Read	Reset (SW)
31:0	**Current Transmit Descriptor Address, lower dword**		yes	value	0
31:0	**Next/Current Transmit Descriptor Address, upper dword** Upper 32 bits for all transmit descriptor addresses in 64-bit addressing mode Should be set once at initialization time		yes	value	0
31:0	**Current Address Counter, lower dword** Incrementing address for bus accesses		yes for tests only	value	0
31:0	**Current Address Counter, upper dword** Incrementing address for bus accesses				
	Current Byte Counter				
31:16	Reserved				
15:0	**Byte Counter**	Decrementing byte counter for bus accesses		value	0
Transmit Queue Control/Status Register					
	Commands				
31:25	Reserved				
24	**Supervisor Idle**	Supervisor SM is in Idle state (see STOP)	ne	value	1
23:22	Reserved				
21	**Reset Desc. Clear**	Set/Clear Reset for Descriptors	exec	0b10	0
20	**Reset Desc. Set**			0b01	1
19	**Reset FIFO Clear**	Set/Clear Reset for FIFOs	exec	0b10	0
18	**Reset FIFO Set**			0b01	1
17	**Run PFI SM**	Reset SM to Idle/Release SM	exec	0b10	0
16	**Reset PFI SM**			0b01	1
15	**Run Supervisor SM**	Reset SM to Idle/Release SM	exec	0b10	0
14	**Reset Supervisor SM**			0b01	1
13	**Run Desc. Read SM**	Reset SM to Idle/Release SM	exec	0b10	0
12	**Reset Desc. Read SM**			0b01	1
11	**Run Desc. Write SM**	Reset SM to Idle/Release SM	exec	0b10	0
10	**Reset Desc. Write SM**			0b01	1
9	**Run Transfer SM**	Reset SM to Idle/Release	exec	0b10	0
8	**Reset Transfer SM**			0b01	1

Bit	Name	Description	Write	Read	Reset (SW)
7	**En Polling On**	Enable/Disable polling of descriptors. If enabled, the periodically generated trigger pulse of the **Descriptor Poll Timer** starts the transfer of this Transmit Queue (same as **Start Tx**). **NOTE:** After initialization or a successfully completed command **Stop Tx,** polling is started with the next command **Start Tx**.	exec	0b10	0
6	**En Polling Off**			0b01	1
5	**Stop Tx**	Stop Transfer of transmit queue Stops further reading of descriptors and transferring of data to the Transmit Queue (clears internal Flag Start TX and forces the Supervisor SM to Idle State). An already running transfer of a transmit frame is completed normally until EOF (including writing back the descriptor(s)), even if more than one descriptor/buffer are consumed. The read value is = 1 until stopping is completed. **NOTE**: If there are not enough descriptors available for transferring until EOF, **Stop Tx** becomes not deasserted. This may be recognized be having **Stop Tx** set while the BMU is running out of descriptors before giving back EOF or having both **Stop Tx** and **Supervisor Idle** set. The only way to get out of this, is to re-issue further descriptors. In case of a transmit queue this **MUST** be avoided by launching complete transmit frames only because of the risk of a transmit underrun.	exec	0/1	0
4	**Start Tx**	Start Transfer of transmit queue	exec	0	0
3	Reserved				
2	**Clr IRQ EOB**	Clear Interrupt **EOB**	exec	0	0

Bit	Name	Description	Write	Read	Reset (SW)
1	**Clr IRQ EOF**	Clear Interrupt **EOF**	exec	0	0
0	**Clr IRQ ERR**	Clear Interrupt **ERR**	exec	0	0
Transmit Queue Flag Register					
31:28	Reserved				
27	**Empty**	FIFO empty	ne	value	
26	**EOF in FIFO**	EOF Flag in FIFO	ne	value	
25	**Watermark reached**	Watermark reached in FIFO	ne	value	
24	Reserved				
23:16	**FIFO Level**	Number of 64-bit words used in FIFO	ne	value	
Transmit Queue Watermark Register					
15:9	Reserved				
10:0	Watermark	Level for requesting data transfer in bytes: Request, if space for n bytes in FIFO.	yes	aw	0x600
Transmit Queue Test Register 1					
	Supervisor SM				
31:28	**Q<3:0>**	States	if **Load**	value	0
27	**Load**	Load States **Q<3:0>** if = 1, **Q<3:0>** are loaded to SM		0	0
26	**Test On**	Testmode ON/OFF	exec	0b10	0
25	**Test Off**			0b01	1
24	**SM Step**	Step SM	exec	0	
	Read Descriptor SM				
23:20	**Q<3:0>**	States	if **Load**	value	0
19	**Load**	Load States **Q<3:0>** if = 1, **Q<3:0>** are loaded to SM		0	0
18	**Test On**	Testmode ON/OFF	exec	0b10	0
17	**Test Off**			0b01	1
16	**SM Step**	Step SM	exec	0	
	Write Descriptor SM				

Bit	Name	Description	Write	Read	Reset (SW)
15:12	**Q<3:0>**	States	if **Load**	value	0
11	**Load**	Load States **Q<3:0>** if = 1, **Q<3:0>** are loaded to SM		0	0
10	**Test On**	Testmode ON/OFF	exec	0b10	0
9	**Test Off**			0b01	1
8	**SM Step**	Step SM	exec	0	
	Transfer SM				
7:4	**Q<3:0>**	States	if **Load**	value	0
3	**Load**	Load States **Q<3:0>** if = 1, **Q<3:0>** are loaded to SM		0	0
2	**Test On**	Testmode ON/OFF	exec	0b10	0
1	**Test Off**			0b01	1
0	**SM Step**	Step SM	exec	0	
	Transmit Queue Test Register 2				
31:8	Reserved				
7	**Add. Counter Test On**	Testmode On/Off	exec	0b10	0
6	**Add. Counter Test Off**			0b01	1
5	**Byte Counter Test On**	Testmode On/Off	exec	0b10	0
4	**Byte Counter Test Off**			0b01	1
3:0	**Counter Step**	Decrement **Current Byte Counter** and **Current Address Counter** by value represented by **Counter Step<3:0>**.	exec	0	0
	Transmit Queue Test Register 3				
31:8	Reserved				
7	Reserved				
6:4	**Mux<2:0>**	Mux position	no	value	0
3	Reserved				
2:0	**VRam<2:0>**	Virtual RAM buffer address	yes	value	0

12.1.33 Receive RAMbuffer Registers

Initialization or re-arrangement of a RAMbuffer should ever start from reset state.

Bit	Name	Description	Write	Read	Reset (private)
	Receive RAMbuffer Start Address				
31:19	Reserved				
18:0	**Start Address**	Start Address in 64-bit words of this queue in external memory. Bit [9:0] are replaced by **Zero**s internally, which means, that the address is limited to multiples of 8kB. Has to be defined after each Reset.	yes (for init/ tests only)	aw	0
	Receive RAMbuffer End Address				
31:19	Reserved				
18:0	**End Address**	End Address in 64-bit words of this queue in external memory. Bit [9:0] are replaced by **One**s internally, which means, that the address is limited to multiples of 8kB. Has to be defined after each Reset.	yes (for init/ tests only)	aw	0
	Receive Buffer Write Pointer				
31:19	Reserved				
18:0	**Write Pointer**	Write Pointer in 64-bit words. Has to be set to **Receive Rambuffer Start Address** after each Reset.	yes (for init/ tests only)	value	0
	Receive RAMbuffer Read Pointer				
31:19	Reserved				
18:0	**Read Pointer**	Read Pointer in 64-bit words Has to be set to **Receive Rambuffer Start Address** after each Reset.	yes (for init/ tests only)	value	0
	Receive RAMbuffer Upper Threshold/Pause Packets				

Bit	Name	Description	Write	Read	Reset (private)
31:19	Reserved				
18:0	**Upper Threshold/ Pause Packets**	If this queue is filled up to this Threshold, signal XmtPausePkt of the related MAC is asserted (if **En Pause** is set). Multiples of 8 bytes. No effect, if set to ZERO.	yes (for init/ tests only)	aw	0
	Receive RAMbuffer Lower Threshold/Pause Packets				
31:19	Reserved				
18:0	**Lower Threshold/ Pause Packets**	Signal XmtPausePkt of the related MAC is deasserted, if the number of bytes is falling below Lower Threshold. Multiples of 8 bytes.	yes (for init/ tests only)	aw	0
	Receive RAMbuffer Upper Threshold/High Priority				
31:19	Reserved				
18:0	**Upper Threshold/High Priority**	If this queue is filled up to this Threshold, all arbiters are granting highest priority to the requests of this queue. Multiples of 8 bytes. No effect, if set to ZERO.	yes (for init/ tests only)	aw	0
	Receive RAMbuffer Lower Threshold/High Priority				
31:19	Reserved				
18:0	**Lower Threshold/High Priority**	All arbiters are granting normal priority to the requests of this queue, if the number of bytes is falling below Lower Threshold. Multiples of 8 bytes.	yes (for init/ tests only)	aw	0
	Receive RAMbuffer Packet Counter				
31:19	Reserved				
18:0	**Packet Counter**	Packet Counter gives the actual number of packets in this queue.	yes (for tests only)	value	0
	Receive RAMbuffer Level				

Bit	Name	Description	Write	Read	Reset (private)
31:19	Reserved				
18:0	**Level**	Level gives the actual number of data in multiples of 8 bytes in this queue (including 16 bytes Statusword per packet).	ne	value	0
	Receive RAMbuffer Control/Test				
		Test			
31:20	Reserved				
19	**Packet Counter Step Down**	Packet Counter decrement	exec	0	
18	**Packet Counter Test On**	Testmode ON/OFF	exec	0b10	0
17	**Packet Counter Test Off**			0b01	1
16	**Packet Counter Step Up**	Packet Counter increment	exec	0	
15	Reserved				
14	**Write Pointer Test On**	Testmode ON/OFF	exec	0b10	0
13	**Write Pointer Test Off**	For Testmode ON this RAMbuffer queue **MUST** be set to NON Operational		0b01	1
12	**Write Pointer Step**	Write Pointer increment	exec	0	
11	Reserved				
10	**Read Pointer Test On**	Testmode ON/OFF	exec	0b10	0
9	**Read Pointer Test Off**	For Testmode ON this RAMbuffer queue **MUST** be set to NON Operational		0b01	1
8	**Read Pointer Step**	Read Pointer decrement	exec	0	
	Control/Status				
7:6	Reserved				

Bit	Name	Description	Write	Read	Reset (private)
5	**St&Fwd On**	Store and Forward On/Off If On, a frame is forwarded from RAMbuffer only, if the complete frame is in RAMbuffer.	exec	0b10	0
4	**St&Fwd Off**			0b01	1
3	**Operational On**	Operational mode ON/OFF OFF is resetting all activities of this RAMbuffer queue for initialization of the pointers and registers	exec	0b10	0
2	**Operational Off**			0b01	1
1	**Reset Clear**	Set/Clear **Reset.** executed, if appropriate bit is set to 1. If **Reset** is set, all Receive RAMbuffer functions and registers are in their reset state.	exec	0b10	0 (SW)
0	**Reset Set**			0b01	1 (SW)

Transmit RAMbuffer Registers

Initialization or re-arrangement of a RAMbuffer should ever start from reset state.

Bit	Name	Description	Write	Read	Reset (private)
	Transmit RAMbuffer Start Address				
31:19	Reserved				
18:0	**Start Address**	Start Address in 64-bit words of this queue in external memory. Bit [9:0] are replaced by **Zero**s internally, which means, that the address is limited to multiples of 8kB. Has to be defined after each Reset.	yes (for init/ tests only)	aw	0
	Transmit RAMbuffer End Address				
31:19	Reserved				

Bit	Name	Description	Write	Read	Reset (private)
18:0	**End Address**	End Address in 64-bit words of this queue in external memory. Bit [9:0] are replaced by **One**s internally, which means, that the address is limited to multiples of 8kB. Has to be defined after each Reset.	yes (for init/ tests only)	aw	0
	Transmit RAMbuffer Write Pointer				
31:19	Reserved				
18:0	**Write Pointer**	Write Pointer in 64-bit words. Has to be set to **Transmit Rambuffer Start Address** after each Reset.	yes (for init/ tests only)	value	0
	Transmit RAMbuffer Read Pointer				
31:19	Reserved				
18:0	**Read Pointer**	Read Pointer in 64-bit words. Has to be set to **Transmit Rambuffer Start Address** after each Reset.	yes (for init/ tests only)	value	0
	Packet Counter				
31:19	Reserved				
18:0	**Packet Counter**	Packet Counter gives the actual number of packets in this queue.	yes (for tests only)	value	0
	Transmit RAMbuffer Level				
31:19	Reserved				
18:0	**Level**	Level gives the actual number of data in multiples of 8 bytes in this queue (including 16 bytes Statusword per packet).	ne	value	0
	RAMbuffer Control/Test				
		Test			
31:20	Reserved				

Bit	Name	Description	Write	Read	Reset (private)
19	**Packet Counter Step Down**	Packet Counter decrement	exec	0	
18	**Packet Counter Test On**	Testmode ON/OFF	exec	0b10	0
17	**Packet Counter Test Off**			0b01	1
16	**Packet Counter Step Up**	Packet Counter increment	exec	0	
15	Reserved				
14	**Write Pointer Test On**	Testmode ON/OFF For Testmode ON this RAMbuffer queue **MUST** be set to NON Operational	exec	0b10	0
13	**Write Pointer Test Off**			0b01	1
12	**Write Pointer Step**	Write Pointer increment	exec	0	
11	Reserved				
10	**Read Pointer Test On**	Testmode ON/OFF For Testmode ON this RAMbuffer queue **MUST** be set to NON Operational	exec	0b10	0
9	**Read Pointer Test Off**			0b01	1
8	**Read Pointer Step**	Read Pointer decrement	exec	0	
	Control/Status				
7	Reserved				
6	Reserved				

Bit	Name	Description	Write	Read	Reset (private)
5	**St&Fwd On**	Store and Forward On/Off	exec	0b10	0
4	**St&Fwd Off**	If On, a frame is forwarded from RAMbuffer only, if the complete frame is in RAMbuffer. If On, this is overwriting the control on a per frame base as defined by the transmit descriptor. Because most PCI systems are not providing Gigabit Ethernet bandwidth, **St&Fwd** should be switched **On** for the Transmit Rambuffer(s).		0b01	1
3	**Operational On**	Operational mode ON/OFF	exec	0b10	0
2	**Operational Off**	OFF is resetting all activities of this RAMbuffer queue for initialization of the pointers and registers		0b01	1
1	**Reset Clear**	Set/Clear **Reset.** executed, if appropriate bit is set to 1.	exec	0b10	0 (SW)
0	**Reset Set**	If **Reset** is set, all Transmit RAMbuffer functions and registers are in their reset state.		0b01	1 (SW)

12.1.34 Receive MAC FIFOs Registers

Initialization or re-arrangement of a MAC FIFO should ever start from reset state.

Bit	Name	Description	Write	Read	Reset (private)
	MAC FIFO End Address				
31:6	Reserved				
5:0	**End Address**	End Address in 64-bit words of this queue in internal dual ported memory. Max = reset value, min = 0x04 Reset value should be used.	yes (for init/ tests only)	aw	0x3f
	MAC FIFO Write Pointer				

Bit	Name	Description	Write	Read	Reset (private)
31:6	Reserved				
5:0	**Write Pointer**	Write Pointer in 64-bit words of this queue in internal dual ported memory.	yes (for tests only)	value	0
	MAC FIFO Read Pointer				
31:6	Reserved				
5:0	**Read Pointer**	Read Pointer in 64-bit words of this queue in internal dual ported memory.	yes (for tests only)	value	0
	MAC FIFO Packet Counter				
31:6	Reserved				
5:0	**Packet Counter**	Packet Counter gives the actual number of packets in this FIFO.	yes (for tests only)	value	0
	MAC FIFO Level				
31:14	Reserved				
13:8	**Shadow Level**	Shadow Level gives the actual number of data in multiples of 8 bytes in this queue (including 16 bytes Statusword per packet).	yes (for tests only)	value	0
7:6	Reserved				
5:0	**Level**	Level gives the actual number of data in multiples of 8 bytes in this queue (including 16 bytes Statusword per packet).	yes (for tests only)	value	0
	Receive MAC Control				

Bit	Name	Description	Write	Read	Reset (private)
31:24	**Rx Timestamp Timeout**	Timeout of Receive Timestamp after receiving Status. **Rx Timestamp Timeout** is active only, if **Time Stamp** is On. Settings for XMACII MUST be consistent. Timeout time is **Rx Timestamp Timeout** Host Clock cycles. On Timeout, an **IRQ No Timestamp** is generated. Reset value should be used.	yes	value	8
23:16	**Rx Status Timeout**	Timeout of Receive Status after **RxPktValid** deasserted. Timeout time is **Rx Status Timeout** + 1 Host Clock cycles. On Timeout, an **IRQ No Status** is generated Reset value should be used.	yes	value	7
15:14	Reserved				
13	**RxRdy Patch On**	If enabled, RxRdy is asserted, if the XMACII is hanging with an invalid signaling (RcvValid & ~RcvStatusValid & ~RxPktValid) This fixes XMACII's (Rev. B2) errata #26. Must be switched OFF in loopback mode. Must be switched OFF for XMACII/Rev. C	exec	0b10	0
12	**RxRdy Patch Off**			0b01	1
11	**RxValid Timing Patch On**	If enabled, RxRdy is delayed by one Host Clock cycle. This fixes XMACII's (Rev. B2) timing requirements. Must be switched OFF in loopback mode. Must be switched ON for XMACII/Rev. C also.	exec	0b10	0
10	**RxValid Timing Patch Off**			0b01	1

Bit	Name	Description	Write	Read	Reset (private)
9	**En Afull On**	Enable forwarding of the related XMACII's signal RxFIFOAlmostFull to the signal XmtPausePkt to XMACII. Must be switched OFF in loopback mode. Settings for XMACII MUST be consistent. Reset value should be used.	exec	0b10	0
8	**En Afull Off**			0b01	1
7	**En Pause On**	Enable forwarding of the signal XmtPausePkt to XMACII see also **Receive Rambuffer Upper/Lower Threshold/Pause Packets** Must be switched OFF in loopback mode. Settings for XMACII MUST be consistent.	exec	0b10	0
6	**En Pause Off**			0b01	1
5	**En Flush/Rx On**	Enable flushing of XMACII's receive FIFO and invalidating a frame which is in progress to be transferred to the Receive RAMbuffer. This is caused by the XMACII asserting RxFIFOError. FlushRxFIFO to XMACII is asserted. If there is a transfer of a frame in progress, this event is treated like the end of a frame: **RX Status** is set to Zero, **RX Status Invalid** is set. After completion FlushRxFIFO to XMACII is deasserted. Must be switched OFF in loopback mode. Must be set ON for XMACII/Rev. B2. Must be set ON for XMACII/Rev. C, if it's intended to receive packets larger than XMACII's Receive FIFO.	exec	0b10	0
4	**En Flush/Rx Off**			0b01	1

Bit	Name	Description	Write	Read	Reset (private)
3	**Time Stamp On**	XMACII is configured for appending Time Stamp to receive frame Must be switched Off with XMACII, Rev. B2 Must be switched Off in loopback mode. Settings for XMACII MUST be consistent.	exec	0b10	0
2	**Time Stamp Off**			0b01	1
1	**Clear IRQ No Timestamp**	Clear No Timestamp on Receive Interrupt	exec	0	
0	**Clear IRQ No Status**	Clear No Status on Receive Interrupt	exec	0	
	Receive MAC FIFO Test				
		Test			
31:20	Reserved				
19	**Packet Counter Step/Down**	Packet Counter decrement	exec	0	
18	**Packet Counter Test On**	Testmode ON/OFF	exec	0b10	0
17	**Packet Counter Test Off**			0b01	1
16	**Packet Counter Step/Up**	Packet Counter increment	exec	0	
15	Reserved				
14	**Write Pointer Test On**	Testmode ON/OFF	exec	0b10	0
13	**Write Pointer Test Off**			0b01	1
12	**Write Pointer Step**	Write Pointer increment	exec	0	
11	Reserved				
10	**Read Pointer Test On**	Testmode ON/OFF	exec	0b10	0
9	**Read Pointer Test Off**			0b01	1

Bit	Name	Description	Write	Read	Reset (private)
8	**Read Pointer Step**	Read Pointer increment	exec	0	
	Receive MAC FIFO Control				
7:4	**Reserved**				
3	**Operational On**	Operational mode ON/OFF	exec	0b10	0
2	**Operational Off**	OFF is resetting all activities of this FIFO for initialization of the pointers and registers		0b01	1
1	**MAC FIFO Reset Clear**	Set/Clear **MAC FIFO Reset.** executed, if appropriate bit is set to 1.	exec	0b10	0 (SW)
0	**MAC FIFO Reset Set**	If **MAC FIFO Reset** is set, all FIFO functions and Registers are in their reset state.		0b01	1 (SW)

12.1.35 Transmit MAC FIFOs Registers

Bit	Name	Description	Write	Read	Reset (SW)
	MAC FIFO End Address				
31:6	Reserved				
5:0	**End Address**	End Address in 64-bit words of this queue in dual ported memory. Max = reset value, min = 0x05 Reset value should be used	yes (for init/ tests only)	aw	0x3f
	MAC FIFO Write Pointer				
31:6	Reserved				
5:0	**Write Pointer**	Write Pointer in 64-bit words of this queue in internal dual ported memory.	yes (for tests only)	value	0
	MAC FIFO Write Shadow Pointer				
31:6	Reserved				

Bit	Name	Description	Write	Read	Reset (SW)
5:0	**Write Shadow Pointer**	Write Shadow Pointer in 64-bit words of this queue in internal dual ported memory.	yes (for tests only)	value	0
	MAC FIFO Read Pointer				
31:6	Reserved				
5:0	**Read Pointer**	Read Pointer in 64-bit words of this queue in internal dual ported memory.	yes (for tests only)	value	0
	MAC FIFO Packet Counter				
31:6	Reserved				
5:0	**Packet Counter**	Packet Counter gives the actual number of packets in this FIFO. Has to be set to **0** after each Reset.	yes (for tests only)	value	0
	MAC FIFO Level				
31:14	Reserved				
13:8	**Shadow Level**	Shadow Level gives the actual number of data in multiples of 8 bytes in this queue (including 16 bytes Statusword per packet). For checking the FIFO's level both **Level** and **Shadow Level** must be checked.	yes (for tests only)	value	0
7:6	Reserved				
5:0	**Level**	Level gives the actual number of data in multiples of 8 bytes in this queue (including 16 bytes Statusword per packet). For checking the FIFO's level both **Level** and **Shadow Level** must be checked.	yes (for tests only)	value	0
	Transmit MAC Control				
31:21	Reserved				
20:16	**Wait after Flush**	Delay for asserting **TxPktValid** after flushing of the XMACII's Transmit FIFO. The delay is **Wait after Flush** + 1 Host Clock cycles. Reset value should be used.	yes	value	4

Bit	Name	Description	Write	Read	Reset (SW)
15	**Clear IRQ Parity Error**	Clear Parity Error Interrupt	exec	0	
14	Reserved				
13	**En Packet Recovery On**	If enabled, recovery time between packets transferred out of the transmit FIFO is increased by 4 Host Clock cycles. Must be switched OFF in loopback mode. This may fix a potential XMACII problem	exec	0b10	0
12	**En Packet Recovery Off**			0b01	1
11	**TxRdy Timing Patch On**	If enabled, TxValid is delayed by one Host Clock cycle. This fixes XMACII's (Rev. B2) timing requirements. Must be switched OFF in loopback mode. Must be switched ON for XMACII/Rev. C also.	exec	0b10	0
10	**TxRdy Timing Patch Off**			0b01	1
9	**Use AlmostFull On**	If enabled, transferring a 32-bit word to XMACII's FIFO is started only, if it's not almost full. This fixes XMACII's errata #18 (Rev. B2). XMACII's (Rev. B2) Transmit High Watermark (afull) has to be set to 3k7 bytes. Must be switched OFF in loopback mode. Must be switched OFF for XMACII/Rev. C	exec	0b10	0
8	**Use AlmostFull Off**			0b01	1
7	**En Wait 4 Empty On**	If enabled, transferring a transmit frame to XMACII's FIFO is started only, if it's empty. This fixes XMACII's errata #28 (Rev. B2) but reduces the transmit rate. XMACII's (Rev. B2) Transmit Low Watermark has to be set to 0. Must be switched OFF in loopback mode. Must be switched OFF for XMACII/Rev. C	exec	0b10	0
6	**En Wait 4 Empty Off**			0b01	1

Bit	Name	Description	Write	Read	Reset (SW)
5	**En Flush/Tx On**	Enable flushing of XMACII's transmit FIFO and a frame which is in progress to be transferred from the Transmit RAMbuffer. This is caused by the XMACII asserting XmtPktError. FlushTxFIFO to XMACII is asserted. If there is a transfer of a frame in progress, transfer is completed normally by the XMACII Interface, but no data are transferred to XMACII. After completion FlushTxFIFO to XMACII is deasserted. Must be switched OFF in loopback mode. Must be switched ON.	exec	0b10	0
4	**En Flush/Tx Off**			0b01	1
3	**Loopback On**	The XMACII Interface may be set to loopback mode for testing purposes. Transmit data are looped to receive data through the I/O pads. This allows to control the dataflow with an LA There is no activity at the related XMACII control signals. The Transmit Status (descriptors) MUST be set to the expected Receive Status (descriptors).	exec	0b10	0
2	**Loopback Off**			0b01	1
1	**XMACII Reset Clear**	Set/Clear **XMACII Reset.** executed, if appropriate bit is set to 1. If **XMACII Reset** is set, the external XMACII and the XMACII Interface are in their reset state. In order to guarantee a minimum pulse width of 31 HOST CLK cycles, clearing **XMACII Reset** is suppressed within 31 HOST CLK cycles after **XMACII Reset Set**. Write cycles to **XMACII Reset Clear** within the 31 HOST CLK recovery time, are terminated with a Target Retry (no impact on software).	exec	0b10	0
0	**XMACII Reset Set**			0b01	1
	MAC FIFO Control/Test				
		Test			

Bit	Name	Description	Write	Read	Reset (SW)
31:23	Reserved				
22	**Write Shadow Pointer Test On**	Testmode ON/OFF	exec	0b10	0
21	**Write Shadow Pointer Test Off**			0b01	1
20	**Write Shadow Pointer Step**	Pointer increment	exec	0	
19	**Packet Counter Step Down**	Packet Counter decrement	exec	0	
18	**Packet Counter Test On**	Testmode ON/OFF	exec	0b10	0
17	**Packet Counter Test Off**			0b01	1
16	**Packet Counter Step Up**	Packet Counter increment	exec	0	
15	Reserved				
14	**Write Pointer Test On**	Testmode ON/OFF	exec	0b10	0
13	**Write Pointer Test Off**			0b01	1
12	**Write Pointer Step**	Pointer increment	exec	0	
11	Reserved				
10	**Read Pointer Test On**	Testmode ON/OFF	exec	0b10	0
9	**Read Pointer Test Off**			0b01	1
8	**Read Pointer Step**	Read Pointer decrement	exec	0	
	Control				
7:4	Reserved				

Bit	Name	Description	Write	Read	Reset (SW)
3	**Operational On**	Operational mode ON/OFF	exec	0b10	0
2	**Operational Off**	OFF is resetting all activities of this FIFO for initialization of the pointers and registers		0b01	1
1	**MAC FIFO Reset Clear**	Set/Clear **MAC FIFO Reset.** executed, if appropriate bit is set to 1.	exec	0b10	0 (SW)
0	**MAC FIFO Reset Set**	If **MAC FIFO Reset** is set, all FIFO functions and registers are in their reset state.		0b01	1 (SW)

12.1.36 MAC Arbiter Registers

Bit	Name	Description	Write	Read	Reset (SW)
	Timeout Init Values				
31:24	**Timeout Init Value Tx2**	Transmit Fifo of XMACII 2 Max = 255, min = 2 Use reset value for XMACII/Rev. B2 Has to be set to **63** after each Reset for XMACII/Rev. C	yes	aw	54
23:16	**Timeout Init Value Tx1**	Transmit Fifo of XMACII 1 Max = 255, min = 2 Use reset value for XMACII/Rev. B2 Has to be set to **63** after each Reset for XMACII/Rev. C	yes	aw	54
15:8	**Timeout Init Value Rx2**	Receive Fifo of XMACII 2 Max = 255, min = 2 Use reset value for XMACII/Rev. B2 Has to be set to **63** after each Reset for XMACII/Rev. C	yes	aw	54

Bit	Name	Description	Write	Read	Reset (SW)
7:0	**Timeout Init Value Rx1**	Receive Fifo of XMACII 1 Max = 255, min = 2 Use reset value for XMACII/Rev. B2 Has to be set to **63** after each Reset for XMACII/Rev. C	yes	aw	54
	Timeout Timers				
31:24	**Timeout Timer Tx2**		yes (for tests only)	value	0
23:16	**Timeout Timer Tx1**		yes (for tests only)	value	0
15:8	**Timeout Timer Rx2**		yes (for tests only)	value	0
7:0	**Timeout Timer Rx1**		yes (for tests only)	value	0
	MAC Arbiter Control/Test				
		Test			
31	**Timeout Tx2**	= 1, if Timeout Timer = 0	ne	value	1
30	**Timeout Timer Tx2 Test On**	Testmode ON/OFF	exec	0b10	0
29	**Timeout Timer Tx2 Test Off**			0b01	1
28	**Timeout Timer Tx2 Step**	Timer decrement	exec	0	
27	**Timeout Tx1**	= 1, if Timeout Timer = 0	ne	value	1

Bit	Name	Description	Write	Read	Reset (SW)
26	**Timeout Timer Tx1 Test On**	Testmode ON/OFF	exec	0b10	0
25	**Timeout Timer Tx1 Test Off**			0b01	1
24	**Timeout Timer Tx1 Step**	Timer decrement	exec	0	
23	**Timeout Rx2**	= 1, if Timeout Timer = 0	ne	value	1
22	**Timeout Timer Rx2 Test On**	Testmode ON/OFF	exec	0b10	0
21	**Timeout Timer Rx2 Test Off**			0b01	1
20	**Timeout Timer Rx2 Step**	Timer decrement	exec	0	
19	**Timeout Rx1**	= 1, if Timeout Timer = 0	ne	value	1
18	**Timeout Timer Rx1 Test On**	Testmode ON/OFF	exec	0b10	0
17	**Timeout Timer Rx1 Test Off**			0b01	1
16	**Timeout Timer Rx1 Step**	Timer decrement	exec	0	
		Control			
15:4	Reserved				
3	**Fast OE On**	Enable Fast Output Enable of XMACII For future versions of XMACII (if its Output Enable Time becomes faster) executed, if appropriate bit is set to 1. Use reset value until allowed at this place.	exec	0b10	0
2	**Fast OE Off**			0b01	1
1	**Reset Clear**	Set/Clear **Reset.** executed, if appropriate bit is set to 1. If **Reset** is set, all MAC Arbiter functions and registers are in their reset state.	exec	0b10	0 (SW)
0	**Reset Set**			0b01	1 (SW)

Bit	Name	Description	Write	Read	Reset (SW)
	Recovery Init Values				
31:24	**Recovery Init Value Tx2**	Recovery time in # of Host Clock cycles between transfers to XMACII's transmit FIFO. Use reset value for XMACII/Rev. B2 Must be set to 0 after each Reset for XMACII/Rev. C	yes	aw	54
23:16	**Recovery Init Value Tx1**	Recovery time in # of Host Clock cycles between transfers to XMACII's transmit FIFO. Use reset value for XMACII/Rev. B2 Must be set to 0 after each Reset for XMACII/Rev. C	yes	aw	54
15:8	**Recovery Init Value Rx2**	Recovery time in # of Host Clock cycles between transfers from XMACII's receive FIFO. Use reset value for XMACII/Rev. B2 Must be set to 0 after each Reset for XMACII/Rev. C	yes	aw	54
7:0	**Recovery Init Value Rx1**	Recovery time in # of Host Clock cycles between transfers from XMACII's receive FIFO. Use reset value for XMACII/Rev. B2 Must be set to 0 after each Reset for XMACII/Rev. C	yes	aw	54
	Recovery Timers				
31:24	**Recovery Timer Tx2**		yes (for tests only)	value	0
23:16	**Recovery Timer Tx1**		yes (for tests only)	value	0

Bit	Name	Description	Write	Read	Reset (SW)
15:8	**Recovery Timer Rx2**		yes (for tests only)	value	0
7:0	**Recovery Timer Rx1**		yes (for tests only)	value	0
	MAC Arbiter Control/Test				
		Test			
31	**Recovery Tx2**	= 1, if Recovery Timer = 0	ne	value	1
30	**Recovery Timer Tx2 Test On**	Testmode ON/OFF	exec	0b10	0
29	**Recovery Timer Tx2 Test Off**			0b01	1
28	**Recovery Timer Tx2 Step**	Timer decrement	exec	0	
27	**Recovery Tx1**	= 1, if Recovery Timer = 0	ne	value	1
26	**Recovery Timer Tx1 Test On**	Testmode ON/OFF	exec	0b10	0
25	**Recovery Timer Tx1 Test Off**			0b01	1
24	**Recovery Timer Tx1 Step**	Timer decrement	exec	0	
23	**Recovery Rx2**	= 1, if Recovery Timer = 0	ne	value	1
22	**Recovery Timer Rx2 Test On**	Testmode ON/OFF	exec	0b10	0
21	**Recovery Timer Rx2 Test Off**			0b01	1
20	**Recovery Timer Rx2 Step**	Timer decrement	exec	0	
19	**Recovery Rx1**	= 1, if Recovery Timer = 0	ne	value	1

Bit	Name	Description	Write	Read	Reset (SW)
18	**Recovery Timer Rx1 Test On**	Testmode ON/OFF	exec	0b10	0
17	**Recovery Timer Rx1 Test Off**			0b01	1
16	**Recovery Timer Rx1 Step**	Timer decrement	exec	0	
		Control			
15:8	Reserved				
7	**En Rec Tx2 On**	Enable Recovery Timer xxx executed, if appropriate bit is set to 1. Must be set to On for XMACII/Rev. B2 Use reset value for XMACII/Rev. C	exec	0b10	0
6	**En Rec Tx2 Off**			0b01	1
5	**En Rec Tx1 On**		exec	0b10	0
4	**En Rec Tx1 Off**			0b01	1
3	**En Rec Rx2 On**		exec	0b10	0
2	**En Rec Rx2 Off**			0b01	1
1	**En Rec Rx1 On**		exec	0b10	0
0	**En Rec Rx1 Off**			0b01	1

12.1.37 XMACII Registers

All XMACII Registers are mapped in a subsequent range on 32-bit word boundaries. The XMACII interface implements write posting and delayed transactions on read accesses. Target reads are retried until the addressed register is synchronized to PCI Clock.

The XMACII's Node Processor Interface is running in 16bit mode.

The offset of the register addresses in the PCI address space are calculated by multiplying the 16-bit XMACII register addresses by 2.

The XMACII Registers can be accessed with any width at 32-bit word boundaries. On the PCI side all accesses to the XMACII are completed normally. Accesses without any Byte Enables asserted or only to the upper 16 bit (neither CBE<1># nor CBE<0># asserted) are treated like accesses to reserved locations (not forwarded to the XMACII). Note that no byte enables are forwarded to the XMACII. Therefore for all write cycles to the XMACII all 16 bits are written.

Bit	Name	Description	Write	Read	Reset (SW)
Example XMACII Register					
31:16	Reserved				
15:0	**XMACII<x>**	XMACII Register<x>	XMACII datasheet		