Implementation of a Gigabit-Link Protocol with the PLX „PCI 9054“ Interface

Matthias Drochner, Willi Erven, Peter Wüstner, Klaus Zwoll
ZEL FZ Jülich

Tino Häupke, Matthias Kirsch
Struck Innovative Systeme GmbH

Version 1.1
Last modification: 5. November 2001

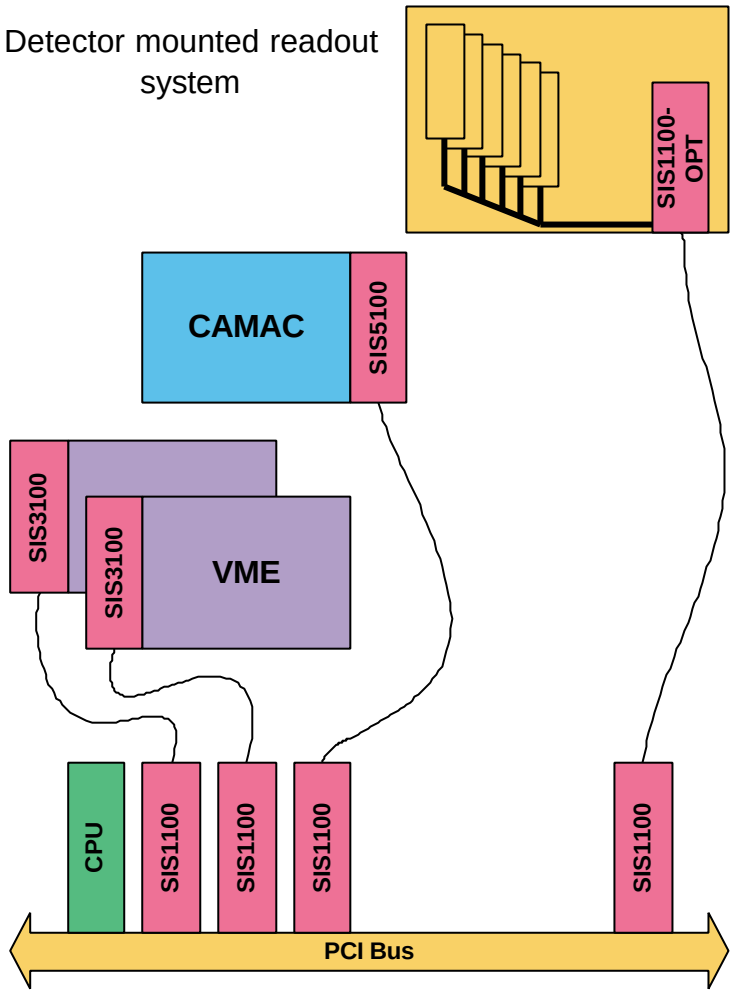
Revision history

Version/ Date	Modification
V1.0 23.01.2001	initial release
V1.1 10.07.2001	Implementation driven modifications. Tranparent mode for test purposes e.g.. Introduction of optical control/status register eingeführt. Combination of various error registers to protocol error register.
V1.1 27.7.2001	extended mailbox registers (mailext) accessible from both sides generation of byte enable bits with direct CPU read
	Translation into English/modifications

Table of contents

1 Introduction

The main focus of the development is the implementation of a high performance PCI/cPCI based event building architecture as illustrated in the graph below. The key component of the system is bidirectional optical data transmission with a data rate of 1 Gbit/s. A point to point architecture was chosen for performance reasons. Data are transferred between a PCI/cPCI module and a dedicated controller. Controller implementations comprise a VME master (SIS3100), a CAMAC/FERA controller (SIS5100) and a universal frontend readout system .



The actual data transfer is based on a protocol that features address based single and block transfers as well as pure data transfers with end of transfer recognition.

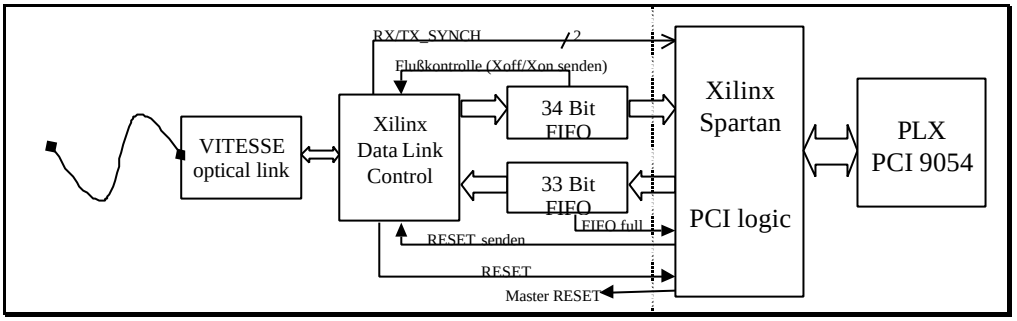
2 Gigabit-Link with PCI 9054

Note:

Throughout the document the expression „word“ denotes a 32-bit word

2.1 Data Link Layer

The hardware was implemented as a two stage design, which is actually a combination of two cards. The first stage is on the SIS1100-OPT Gigabit CMC card. It is in charge of flow control and conditioning of the 32-bit words and consist of the SFF link medium, the SERDES (serializer/deserializer chip), and FPGA and decoupling send/transmit FIFO chips. The second stage is the SIS1100-CMC carrier board. It is in charge of protocol handling and PCI bus interfacing through the PLX PCI to local bus bridge chip 9054.



Simplified block diagram

2.1.1 Special Characters

Following special characters are used for data link control:

Char	Code	Name	Function
K28.0	0x1C	SC_PROT	Signature for protocol header, transmitted as first Byte of the Header Word (special word).
K28.1	0x3C	SC_RESET	Reset Signal (Trouble Reset)
K28.2	0x5C	SC_XOFF	Flow control (FIFO almost full)
K28.3	0x7C	SC_XON	Flow control (FIFO empty)
K28.4	0x9C	SC_RXERR	Receive Error (no receiver synchronisation)

2.1.2 Synchronisation

Proper link connection to the remote station is detected by the *DataLink Control* by the receiption of Idle Characters K28.5. The RX_SYNCH signal flags the conditione. SC_RXERR Special Characters will be send by the *DataLink Control* in a cyclic manner if receiption of Idle Characters is disturbed. The TX_SYNCH will be tied to 0 by the *DataLink Control* as long as SC_RXERR are received. TX_SYNCH will be set to 1 as soon as no SC_RXERR Character is recived for more than 16 idle characters. Both signals (RX_SYNCH , TX_SYNCH) are part of the status register.

2.1.3 Flow Control

The *Data Link Control* sends a SC_XOFF special character as soon as the input FIFO reaches the almost full condition. A SC_XON special character is send as soon as the FIFO fill level will go back below almost full. No further data are transmitted from the output FIFO once the *DataLink Control* is in Xoff state.). Upon recognition of the FIFO full the PCI logic will insert wait states during a block transfer. After a number of wait states (corresponding to some 50µs) the transfer will be terminated and the corresponding error condition will be set. The driver has to handle this condition with a master reset.

2.1.4 Master reset

A master reset can be generated by sending the special character SP_RESET from one side, if the other side is hung/locked.

2.2 PCI Logic

The PLX chip has two programmable address windows for direct PCI Slave access (**SPACE0** and **SPACE1**). In addition two DMA channels (**DMA0** and **DMA1**) can be used in a universal manner.

SPACE0 is used for registers. The size is 4kB festgelegt, the range from 0x000-0x77C is used for the *Local Register* set and the range from 0x800-0xFFC for the *Remote Register* set.

SPACE1 is used for direct memory access on the remote side. The size is currently 4 MB, an extension to **256 MB** is under consideration however. The size can be changed by a reconfiguration of the EEPROM, at the same time firmware changes will be required too in addition.

DMA0 this channel is operated in demand mode im and is used to read from a remote data stream only (readout data e.g.). It is initialized and remains open until the next data stream with end marker was received (interrupt). A *Local Register*, which holds the transfer length is implemented (***d0_bc***).

DMA1 used for a direct block transfer to the remote station. Memory mapping tables (scatter gather table) are generated in host system memory (under control of the driver).

Note:

Don't mix up the expressions **SPACE0** and **SPACE1** with the content of the Space Byte in the protocol header.

The PLX chip is configured in a fashion, that a **SPACE0**, **SPACE1** or **DMA** (block transfer) access can be identified by the upper bits (A[31:30]) of the local address. A **DMA0** transfer is recognized by the DREQ0/DACK0 signals. Following coding is used:

	LA31	LA30
DMA0 DMA1	**0**	**x**
SPACE0	**1**	**0**
SPACE1	**1**	**1**

The local address, i.e. the address on the add on side of the PLX chips, is configured by various mapping registers. The PCI side address space (**SPACE0** and **SPACE1**) is mapped to a fixed local side address space. The local address during a DMA transfer is defined in a DMA descriptor, which can reside in the PLX chip or in system memory . Both DMA channels have to be initialized in a fashion that allows termination by an external EOT signal.

2.2.1 Timeout

Timeouts are a critical issue. A PC/PCI ⇔ PC/PCI connection is considered for the following. The "PCI Bus Latency Timer" determines the time a PCI master can hold the PCI bus on a PCI bus access. This value is set to 1.9µs bz the PCI BIOS normally, can be increased up to 7.7µs however. If it is assumed, that up to 5 PCI masters can deploy this value, a worst case wait time of 38µs for a master to get PCI busmastership results. In a real life environment up to 5µs were obtained with the PLX bridge chip (what is unexpected low).

A window for allowed addresses for "local to PCI" access is set by the driver in the PLX bridge chip. If the remote side tries to access an address outside of the allowed window the timeout will occur. If the address is valid however, and the timeout occurs before the bus cycle was terminated, an undefined state of the PLX bridge chip will result, that can be resolved by a master reset of the chip only. This is in favour for a fairly long timeout value.

The master timeout (**Mtout**) was defined to be **approx. 490µs**. The other timeout values have values, that differ by 2µs steps of the implemented prescaler, for proper error detection/recognition.

The timeout value for a "local to PCI" cycle is set to **Mtout-3*2µs**.

The protocol timeout for a direct erad request is **Mtout-1*2µs**. The general „PCI to local" timeout (driver access to a non existing address e.g.) is **Mtout**.

Protocol receiption timing is monitored also. For single transfer protocols and block transfer header (AM, address, byte count), the maximum gap between individual words has to be less than **approx. 480ns**. Data words can have a spacing of up to **Mtout** during a block transfer.

Send process timing monitoring is related to the fill level of the output FIFO. Transmission of a protocol is not started unless the output FIFO can hold at least 128 words (almost full flag). A timeout error will incurr if the time **Mtout-1*2µs** is exceeded. It has to be kept in ind, that the total time for the send process is limited to this timeout, i.e. the time which is available until the read confirmation is received is shorter by the time that was spend on the wait for the required FIFO space to become available.

Direct PCI BUS access

The PCI bus can be accessed by the remote side directly, in this case the PCI memory address is defined by the remot side. The access can be indirect (protocol driven) by the remote side however also, in this case the PCI address is given by the driver through the *pipelined read buffer* (***rd_pipe_buf***) or the *EOT byte count buffer* (***d0_bc_buf***).

All addresses under consideration have to be allowed by the PLX bridge chip. The PLX registers ***dmrr***, ***dmlbam*** and ***dmpbam*** have to be set by the driver accordingly, as shown in the example below, to define an address window.

```
//
//--------------------------- set_initiator_window --------------------------
//
void set_initiator_window(
   int   width,   // window width in bits
                  // 0: disable local to PCI access
   int   base)    // base address
{
   u_long   msk;

   if (width == 0) {
      plx_regs->dmpbam=0;              // disable local to PCI access
      return;
   }

   if (width < 16) width=16;           // min width is 16
   msk=0xFFFFFFFF <<width;

   plx_regs->dmrr  =msk;
   plx_regs->dmlbam=base &msk;
   plx_regs->dmpbam=(base &msk) |0x1;  // enable local to PCI access
}
//
```

If the address window is not set, or the PCI address does not fall within this window, the PLX bridge chip will ignore the Master Request, which will be terminated by the local Master-Timeout in return (**Mtout**).

The request (indication) is answered with the **PE_BERR** error confirmation (reply) in the case of a direct access by the remote side. The bit **PROT_ERR** will be set in the status register.

Attention!

The *receive state machine* hangs up if the *pipelined read buffer* address (***rd_pipe_buf***) or the *EOT byte count buffer* address (***d0_bc_buf***) are not allowed. This condition is flagged bz the protocol error register ***prot_error*** showing the error condition **E_DLOCK** constantly, and the balance counter ***p_balance*** staying constant. This implies, that the driver has to guarantee, that buffer addresses are valid always.

2.2.2 Deadlock handling

A dead lock situation can be caused by both sides starting a transfer transaction at the same time. kommen. Leider läßt sich diese Situation entgegen der ursprünglichen Annahme nicht lösen. Ein Deadlock kann dann entstehen, wenn der Treiber eine Leseoperation startet und protokollbedingt auf das Ergebnis warten muß (z.B. bei einem Lesezugriff auf die remote Seite oder auf das eigene **prot_error** Register wenn das letzte confirmation noch nicht eingetroffen ist. Wenn zur gleichen Zeit ein indication empfangen wird, welches auf den PLX Chip zugreift (auf PLX Register oder PCI BUS) kommt es zum Deadlock. In diesem Fall wird der Lesezugriff des Treibers beendet und ein Deadlock-Fehler gemeldet. Wenn der Treiber gerade das **prot_error** Register gelesen hat, kann er diese Aktion einfach wiederholen. Wenn der Treiber aber ein remote read ausgeführt hat, muß es schon anschließend das **prot_error** Register lesen, um diese Situation zu erkennen. Das gelesene Datenwort geht dabei nicht verloren, sondern wird im Register <tc_dal> abgelegt.

Der Schreibzyklus wird durch ein Post Write entschärft. D.h., ein Schreibzyklus wird sofort abgeschlossen, sobald das Protokoll gesendet ist. Das Confirmation wird nicht abgewartet. Ein nachfolgender Lesezyklus auf das **prot_error** Register wird solange ausgedehnt, bis das Confirmation eingetroffen ist, damit ein Fehler zum richtigen Zeitpunkt erkannt wird. The previously mentioned timeout value **Mtout-1*2µs**. holds here as well.

2.3 Register

All registers are mapped into **SPACE0** Is is subdivided into two areas of identical size for the *Local* und *Remote register sets*. Auf alle Register kann nur im 32 Bit Mode zugegriffen werden. Access to non existing local registers will be confirmed with *local bus error* (see status register). It is assumed, that the driver writes valid data into the registers. A senseless protocol can result, as no data checking is performed. The mailbox registers and the doorbell register can be written to by the remote side, the first group can be accessed by the remote side partially.

Offset	Name	Function	local access	remote access	MR
0x00	ident	Type-Identifier/Version	RO	RO	
0x04	sr	Status Register	RD selective clear	RD selective set/clr	(0)
0x08	cr	Control Register	RD selective set/clr	RO	0
0x0C	semaphore	semaphore, treated in a special manner by hardware	RW	RO	0
0x10	doorbell	Interrupt generation, mapped to PLX register L2PDBELL	RD selective clear	WO selective set	--
0x14-0x1C	unused				
0x20-0x3C	mailbox[8]	8 Mailbox register, mapped to PLX Registers MBOX[0-7].	RW	WO	--
0x40-0x7C	unused				

The next group can be accessed locally only. The remote side will read zero on access to these (or non implemented) registers.

Offset	Context	Name	Function	MR
0x80	Temporary iransfer. I.e. *pipelined read* and block read Page **34**	t_hdr	.sc controls protocol start	--
0x84		t_am	[15:0] VME Address Modifier	--
0x88		t_adl	address word low	--
0x8C		t_adh	address word high	--
0x90		t_dal	data word low	--
0x94		t_dah	data word high	--
0x98	unused			
0x9C	Temporary confimation	tc_hdr	RO and clear: last read confimation (temporary read or deadlock)	0
0xA0		tc_dal	RO: data low	0
0xA4		tc_dah	RO: if tc_hdr.be[7:4] non 0 only not yet implemented	0

0xA8	Error indication	p_balance	[11:0] Counter for missing confirmation, reset by write access	0
0xAC		prot_error	[9:0] protocol error b7-0: error code b8: local error, protocol not send or no confirmation b9: remote error (error confirmation)	0
0xB0	DMA0	d0_bc	[19:0] byte count for DMA0 space (DAQ mode)	0
0xB4		d0_bc_buf	phys. address for byte count buffer, [0] is enable bit	0
0xB8		d0_bc_blen	[13:0] as above, length, [1:0] =0	0
0xBC	DMA1 block transfer or *pipelined read*	d_hdr		0
0xC0		d_am	[15:0] VME Address Modifier	0
0xC4		d_adl	bit 31 only	0
0xC8		d_adh		0
0xCC		d_bc	[19:0] Byte Count, [1:0] =0	0
0xD0-0xD4	unused			
0xD8	*pipelined read*	rd_pipe_buf	phys. address for *pipelined read* data	0
0xDC		rd_pipe_blen	[13:0] as above, length, [1:0] =0	0
0xE0-0xE4	unused			
0xE8	Transparent test register for transparent mode only	tp_special	WR: Write of „special data“. RD: Read of „special data“ from input-FIFO.	--
0xEC		tp_data	WR: Write of normal data. RD: Read of normal data from input-FIFO.	--
0xF0	Optical control/status	opt_csr	for test of the optical link module Piggy back.	(0)
0xF4		jtag_csr	JTAG register, for FPGA FLASH programming	(0)
0xF8-0xFC	unused			
0x100-0x3FC	Extended Mailbox	mailext [192]	local: RW remote: RW	--

0x400	*SPACE1* Descriptor descr[0]	.hdr		--
		.am	[15:0] VME Address Modifier	--
		.adl	[31:16] for the time being for 4MB [31:22] later for 256MB	--
		.adh		--
0x410-0x7FC	descr[1..63]		63 additional descriptors	--

The **SPACE1** descriptors are mapped to address 0x400 to facilitate access. The block transfer length is specified in bytes, bit 0 and 1 are ignored however (i.e. 32-bit words are transferred).

2.3.1 ident: Identifier/Version

This register has to be present in all implementations. It has offset 0 and is read only.

BIT	**access**	**Name**	Function
7-0 000000FF	RO	Identifier	1 = PCI/PLX interface 2 = VME controller 3 = CAMAC/FERA controller 4 = Readout system with LVD SCSI
15-8 0000FF00	RO	Hardware Version	1..255
23-16 00FF0000	RO	Firmware Identifier	1 = universal others for special firmware
31-24 FF000000	RO	Firmware Version	1..255

2.3.2 sr: Status Register

This register holds state and event bits. It can be read by both sides. The remote side has selective set/clear access to part of the bits. All bits are 0 upon reset. Bits that are WO on remote access are write only and will be read as 0.

BIT	Name	local acc	remote acc	Function
0 00000001	RX_SYNCH	RO	RO	Optical receiver synchronized
1 00000002	TX_SYNCH	RO	RO	Remote optical receiver synchronized
2 00000004	INHIBIT	RO	RD WR: sel set	Data transfer to remote side blocked (register access except)
3 00000008	CONFIGURED	RO	RD WR: sel set	allows RESET or Power Up recognition on remote side
4 00000010	SYNCH_CHG	RD WR: sel clr	RO	RX/TX_SYNCH changed
5 00000020	INH_CHG	RD WR: sel clr	RO	INHIBIT changed
6 00000040	SEMA_CHG	RD WR: sel clr	RO	Semaphore was changed by remote station
7 00000080	REC_VIOLATION	RD WR: sel clr	RO	Transmission error (optical receiver), indicates bad connection.
8 00000100	RESET_REQ	RD WR: sel clr	RO	SP_RESET Special Character received
9 00000200	DMA_EOT	RD WR: sel clr	RO	**EOT** for DMA0 detected
10 00000400	MBX0	RD WR: sel clr	RO	Mailbox Register 0 was written to
11 00000800	S_XOFF	RD WR: sel clr	RO	Status XOFF, das ouput FIFO full or input FIFO can not be emptied
12 00001000	LEMO_IN_0_CHG	RD WR: sel clr	RO	Change on LEMO Input Signal 0.
13 00002000	LEMO_IN_1_CHG	RD WR: sel clr	RO	Change on LEMO Input Signal 1.
14 00004000	PROT_END	RD WR: sel clr	RO	Protocol End, generated on p_balance null or with prot_error being set
15 00008000	PROT_L_ERR	RD WR: sel clr	RO: 0	Protocol error caused by local request

Bits 16 to 31 not be used as interrupt

16 00010000	DMA0_BLOCKED	RO	RO: 0	DMA0 not ready or ***d0_bc_blen*** is 0.
17 00020000	NO_PREAD_BUF	RO	RO: 0	Length of *pipelined read buffers* is 0
18 00040000	PROT_ERR inhibit	RD WR: sel clr	RD: 0 WO: sel clr	Protocol error caused by remote side
19 00080000	BUS_TOUT configured	RD WR: sel clr	RD: 0 WO: sel clr	Bus access on access to non existing (valid) register
20 00100000	TP_SPECIAL	RO	RO: 0	"special word" present
21 00200000	TP_DATA	RO	RO: 0	"data word" present
31-22 FFC00000		RO: 0	RO: 0	

Bits that are hatched in grey will generate an interrupt if the corresponding enable bit is set in the control register.

RX_SYNCH valid characters are seen by the optical receiver from remote station, i.e. the connection between remote sender and local receiver is stable.

TX_SYNCH is set when both directions of the optical connection are stable. This implies, that , RX_SYNCH is set also.

CONFIGURED this bit assists the remote side to check, whether the counterpart is still configured, or whether it was restarted by a reset. The bit is cleared by a reset and can be changed by the remote side only. A LED should be connected to this bit for optical control where possible.

REC_VIOLATION this bit is set whenever the data link layer reports a receive error for a word from the FIFO (bit-33). The error message is used for book keeping only. A protocol error will result for the current protocol, as error prown words are not latched (see **PROT_ERR**).

This error message is not implemented on the optical data link piggy back yet.

RESET_REQ receiption of a SC_RESET special character is displayed. It can be set by the remote side, and can be generated by the SIS1100-OPT data link piggy back with resynchronisation following a loss of synchronisation (see RX_SYNCH and TX_SYNCH also).

DMA_EOT this bit will be set if a block transfer is terminated with the **END** special word, that has **EOT** signature. This holds for indication and confirmation.

MBX0 the remote side has written to the mailbox register mailext[0]. The mailbox registers within the PLX bridge chip (mailbox[7:0]) can generate an interrupt.

S_XOFF receiption of data is blocked for one of the following reasons:

- the length of the PIPE-buffer (rd_pipe_blen) is zero,
- the demand DMA0 channel is inactive
- the length of the *EOT byte count buffer* (d0_bc_blen) is zero.

One of the two status bits NO_PREAD_BUF or DMA0_BLOCKED is set.

PROT_ERR this status bit will be set with following conditions:

- receiption of out of protocol data
- receiption of incomplete protocol (refer to timeout page **5)**
- invalid indication (special word)
- bus error on PCI memory access (invalid address)
- invalid confirmation

Which of the conditions caused the error can not be resolved. Diese Fehlermeldung soll nur der Statistik dienen.

BUS_TOUT set after a local timeout. Caused by access to non existing address. Holds for access to registers **tp_special** and **tp_data** also if transparent mode (TRANSPARENT) was not activated.

PROT_END set when all pending confirmations have been received (register p_balance reaches zero) or an error was detected.

DMA0_BLOCKED set if receiption blocked because:

- DMA0 channel not ready
- length of *EOT bytecount buffers* (***d0_bc_blen***) is zero

Cleared as soon as the cause is no longer true. The driver has dump the data by emptying the FIFO in transparent mode if it will not want to use the data.

NO_PREAD_BUF set when receiption is blocked due to a zero length PIPE buffer. (rd_pipe_blen) null ist. Cleared as soon as the cause is no longer true. As in the **DMA0_BLOCKED** case, the driver has dump the data by emptying the FIFO in transparent mode if it will not want to use the data.

FIFO_FULL/FIFO_PARFULL
the output FIFO is full, almost full respective, because the remote side is blocking. This condition may be resolved by a reset command (Bit1 control register) to the remote side. This error condition should be checked whenever a protocoll timeout error (*LE_TO*) has occured.

TP_SPECIAL the special word receive register contains a valid character. It can be read from the register **tp_special** in transparent mode.

TP_DATA the data word receive register contains a valid character. It can be read from the register **tp_data** in transparent mode.

2.3.3 cr: Control Register

The control register holds general control functions and the interrupt enable bits. All bits are implemented in a J/K style, i.e. they are selective set, if the corresponding bit (in the first half og the register [15:0]) is written to with a 1 and selective clear if the corresponding bits in the second half ([31:16]) is written to with a 1 sind. Writing a 0 to a bit has no effect, an undefined toggle state may result if 1 is written both to the on and off bit (bit 0 and 16 e.g.). The control register is read only for the remote side, the reset state is all bits at 0.

BIT	Name	local acc	
0 00000001	RESET	RD: 0 WR: sg shot	single shot, Master Reset
1 00000002	TRANSPARENT	RD: WR: sel set	data can be written to and read from the FIFOs directly in transparent mode (test feature).
2 00000004	READY	RD: WR: sel set	Data transfer by remote side enabled (register access is possible always).
3 00000008	BIGENDIAN	RD: WR: sel set	all 32-bit data words will be swapped
15-4 00007FF0	INT_ENABLE	RD: WR: sel set	Interrupt enable for bit 4 to 15 in status register.
16 00010000	REM_RESET	RD: 0 WR: sg shot	single shot, sends reset signal to remote side (SP_RESET special character).
17 00020000	transparent	RD: 0 WR: sel clr	selective clear Bit 1
18 00040000	ready	RD: 0 WR: sel clr	selective clear Bit 2
19 00080000	bigendian	RD: 0 WR: sel clr	selective clear bit 3
31-20 7FF00000	int_enable	RD: 0 WR: sel clr	selective clear bit 4 to 15

RESET a reset is generated by writting a 1 to this bit. Other reset conditions are reset on the PCI bus and software reset of the PLX bridge chip by the driver. A reset sets all control register bits to 0, clears the input and output FIFO as well as the data link layer.

REM_RESET a SP_RESET special character is send to the remote side if this bit is written to with a 1. The remote side has to react by initializing ist logic to a defined state.

READY all non register access protocols from the remote side are blocked as long as this bit is not set to avoid memory access induced undefined errors as long as the local side is not initialized yet.

BIGENDIAN data words are swapped during a simple request (non block transfer) if this bit is set. The bit has no effect for indications. Data can be swapped by the PLX bridge chip during a block transfer in addition by the driver.

TRANSPARENT

"protocol handling" is disabled if this bit is set. Data can be written to the output FIFO and read from the input FIFO in a transparant fashion. The register **tp_special** is used for "special data" and the register **tp_data** for normal data. Transparent mode was implemented for test purposes only.

2.3.4 semaphore: Semaphore Register

It is under consideration to implement firmware designs, that allow the remote side to initiate transfer activity without local request. The semaphore register was implemented to arbitrate for link mastership. The protocol handles individual treatment for the local and remote semaphore register. Both registers are preset to 0. This reset state, which will be recovered after a master releases the link also. No direct write access to the *remote semaphore register* is implemented. A semaphore register can have the values 0, 1 and 2 only. The content of the local and the remote register are complementary, i.e. if one of the two registers is at 1 the other will be at 2. The station is link master if the local semaphore register has a value of 1.

Following steps are required to obtain link mastership:

1. write 1 to *local semaphore register*.
2. read *local semaphore register* (after confirmation). The transaction was successful if the content is 1, if the content is 2, the remote station is (still) master.
3. In case of conflict the value will remain at 0 ist. The driver should retry the request after a short (possibly random length) delay.

Bit 0 is decoded on write access only. A value of 0 indicates release, a value of 1 request. Remote station write access to the semaphore register is blocked for the rest of the transaction and confirmed accordingly. The transaction continues with a write access to the remote semaphore register . The Local semaphore register content will be set according to the status of the arbitration process as last step of the transaction.

The request is answered by error reply (confirmation) for ease of implementation. The confirmation codes are listed below:

0x80 request ist was executed on remote side

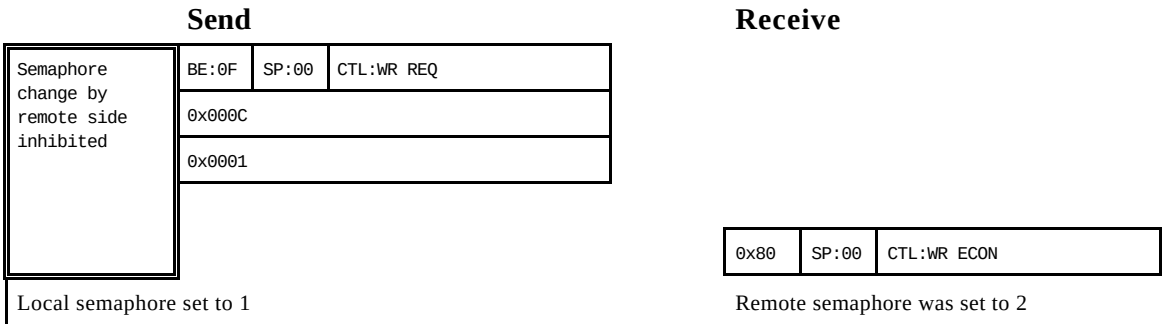
0x81 remote side is already master. Semaphore register will be set to 2 on local side

0x82,0x83 conflict situation. Local Semaphore register remains unchanged

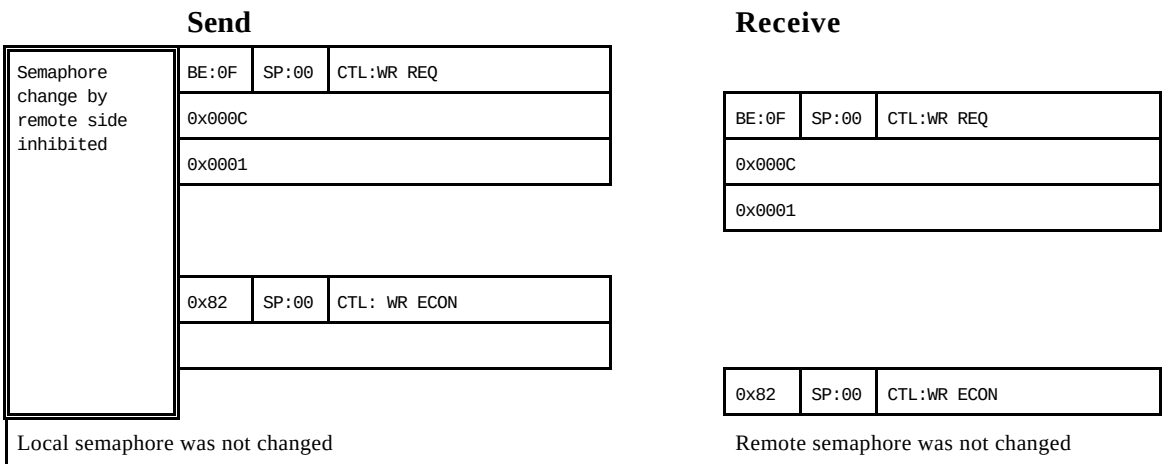
Interrupt generation upon *Semaphore register* content change by the reomote station can be activated (see control/status register bit 5).

A remote side without *Semaphore register* implementation has to send an error reply with code 0x80 upon write access to the address.

Write command to local semaphore with 1



Write command to local semaphore with 1 — same on remote side



2.3.5 doorbell: Interrupt Register

The register ***doorbell*** is associated with the PLX register L2PDBELL. On remote station write access the set bits (selective set). Interrupt generation can be connected to the L2PDBELL non 0 state. ist, kann ein Interrupt ausgelöst werden. On PCI side (driver) write access set bits of the datum will be cleared (selective clear).

2.3.6 mailbox: Mailbox Register

Die 8 Mailbox Regiser ***mailbox[0]*** bis ***mailbox[7]*** befinden sich in den PLX Registern MBOX0 bis MBOX7. Sie können von der remote Seite nur beschrieben werden. Das Auslösen eines Interrupts ist nicht möglich.

On remote station write access to ***mailbox[0]*** status register bit ***MBX0***, which can be used for interrupt generation, will be set.

2.3.7 mailext: Extended Mailbox register

192 extended mailbox registers, that can be read from and written to by both sides, are implemented. The status bit **MBX0** is set when the remote side writes to the first extended mailbox register (mailext[0]). The condition can be used for interrupt generation also.

2.3.8 opt_csr: optical control/status register

BIT	Name	local acc	Function
0 00000001	VSC_SELF_TEST	RW	
1 00000002	SEND_XOFF_LEVEL	RW	
2 00000004	TRANSMIT_LINK_WAIT	RW	
3 00000008		RW	frei
4 00000010	LEMO_OUT_0	RW	control of LEMO output 0
5 00000020	LEMO_OUT_1	RW	control of LEMO output 1
6 00000040	LED_0	RW	control of right lower LED
7 00000080	LED_1	RW	control of rechten upper LED
8 00000100	LEMO_IN_0	RO	state of LEMO input 0
9 00000200	LEMO_IN_1	RO	state of LEMO input 1
15-6 0000FC00		RO: 0	unsused
16 00010000	TRANSMIT_LINK_WAIT_FLAG	RD WR: sel clr	latching, to be cleared by write access
17 00020000	GET_XON_FLAG	RO	
18 00040000	LWORD_UNLIGNED	RO WR: sel clr	latching, to be cleared by write access
19 00080000		RO: 0	unused
20 00100000	TBERR	RO WR: sel clr	latching, to be cleared by write access
21 00200000	UORUN_ERROR	RO WR: sel clr	latching, to be cleared by write access
22 00400000	DISPAR_ERROR	RO WR: sel clr	latching, to be cleared by write access
23 00800000	BAND_ERROR	RD WR: sel clr	latching, to be cleared by write access
27-24 0F000000	BUSMODE	RO	ID from optical piggy back
28 10000000	FIFO_PEF	RO	FIFO partial empty Flag
29 20000000	FIFO_EF	RO	FIFO empty Flag
30 40000000	S_FIFO_PFF	RD WR: sel clr	FIFO partial full Flag latching, to be cleared by write access
31 80000000	S_FIFO_FF	RD WR: sel clr	FIFO full flag latching, to be cleared by write access

The optical control/status register hold signals, that are on the CMC connector of the optical piggy back card (SIS1100-OPT). Most were implemented for debugging during the implementation phase. Following bits can be used for debugging/test purposes and possibly synchronization of several PCI systems (LEMO signals, as far as I/O option is installed).

LED_0 LED_1

two user LED's on the SIS1100-OPT gigabit link CMC can be switched on and off through these bits.. The user LEDs are especially helpful to check initial access to the SIS1100 during driver development or for test/debug purposes.

LEMO_OUT_0, LEMO_OUT_1

two LEMO TTL outputs on the SIS1100-OPT gigabit link CMC can be switched on and off through these bits if the I/O option is installed.

LEMO_IN_0, LEMO_IN_1

the bits reflect the current state of the two LEMO inputs on the SIS1100-OPT (if the I/O option is installed). A level change on the inputs (0 to 1 or vice versa) sets the corresponding bit in the status register (**LEMO_IN_0_CHG**, **LEMO_IN_1_CHG**). They can generate an interrupt and are selective clear.

2.3.9 p_balance, prot_error: general error flagging

Der Zähler **p_balance** wird mit jedem request inkrementiert und mit jedem confirmation dekrementiert. Er muß normalerweise also immer null sein. Er kann durch einen Schreibbefehl mit einem beliebigen Wert auf null gesetzt werden. Im Register **prot_error** wird jeglicher Protokoll-Fehler angezeigt. Es wird immer nur der erste Fehler gespeichert. Nach dem Lesen wird dieses Register auf null gesetzt, so daß der nächste Protokoll-Fehler wieder gespeichert werden kann. Wenn der Inhalt des Registers null ist und der balance counter ungleich null, wird der Lesevorgang verzögert, bis der Zähler **p_balance** null ist. Nach einem Timeout von **Mtout-1*2µs** (siehe Seite **5**) wird die Kennung *LE_TO* zurückgegeben. In diesem Fall muß auch das Register **p_balance** durch einen Schreibbefehl auf 0 gesetzt werden. Sollte es beim Lesen zu einem deadlock kommen, wird die Fehlerkennung *E_DLOCK* zurückgegeben ohne den Wert in **prot_error** zu verändern. Der Treiber kann das Lesen so lange wiederholen, bis die Fehlerkennung ungleich *E_DLOCK* ist.

Bit 8 und 9 des **prot_error** Registers haben eine besondere Bedeutung. Bit 8 ist gesetzt, wenn der Fehler eine lokale Ursache hat. D.h. der request konnte nicht gesendet werden oder es wurde kein confirmation empfangen. Bit 9 ist gesetzt, wenn ein error confirmation empfangen wurde. In diesem Fall wird der error code (8 Bit) aus dem error confirmation übernommen. In folgender Tabelle sind die generellen Fehler aufgeführt. Wenn Bit 8 gesetzt ist, ist ein L (local) vorgesetzt und wenn Bit 9 gesetzt ist, ist ein R (remote) vorgesetzt. Die aufgeführten RE_... Fehler beziehen sich auf dieses Interface als remote Station (PC/PCI ⇔ PC/PCI Kopplung).

Name	Code	Bedeutung
E_DLOCK	0x005	Deadlock as preliminary error on read of **prot_error** register
LE_SYNCH	0x101	Transmission error, protocol could not be send due to lack synchronisation (see status TX_SYNCH).
LE_NRDY	0x102	protocol lock (INHIBIT) set by remote station
LE_XOFF	0x103	output FIFO almost full
LE_RESOURCE	0x104	receiption blocked, because of lack of PIPE-buffer or DMA0 buffer (see **S_XOFF** status). The error condition is flagged until the cause is resolved
LE_DLOCK	0x105	a direct read access could not be completed due to a deadlock The data are available in the registers tc_hdr/tc_dal however.
LE_TO	0x107	Protocol timeout, request not answered. The error is flagged until **p_balance** reaches zero (or is reset).
RE_NRDY	0x202	die remote station ist nicht ready (control register **READY**)
RE_PROT	0x206	Protocol error, protocol could not be executed (wrong request e.g.)
RE_TO	0x207	Protocol timeout, ein indication wurde nicht komplett empfangen (z.B. wegen einer Übertragungsstörung).
RE_BERR	0x208	bus error, address not valid or not present
RE_FERR	0x209	FIFO error, at end of block transfer e.g.

2.3.10 tc_hdr, tc_dal: temporary confirmation

The registers have two functions. The first function is to store the result of a simple read access via the temporary protocol. The second function is to store the result of a direct read access if it was terminated by a simple read access. The register **tc_hdr** is cleared on read access. If the error condition *LE_DLOCK* was detected after a read access , a potential error message could not be stored by the remote side, because the error register **prot_error** was occupied with the error condition. . The header word of the confirmation, that is stored in **tc_hdr** has to be checked in this case.

Even after a timeout condition (*LE_TO*) the response is not lost if the confirmation is received at a later point in time. The program example below shows a simple read access with all possible error conditions.

```
int direct_read(        // return error code
   u_int32  *rem_addr,  //
   u_int32  *data)      // return data read
{
   u_int32  perr;

   *data =*rem_addr;
   while ((perr =lreg->prot_error) == E_DLOCK);   // temporary deadlock

   if (perr == 0) return 0;                       // no protocol error

   if ((perr != LE_DLOCK)/* && (perr != LE_TO)*/) return perr;

   while ((perr =lreg->prot_error) == E_DLOCK);
   if (perr != 0) return perr;

   hdr =lreg->tc_hdr;
   if ((hdr &0x0300) == ECON) return 0x200|(hdr >>24);

   *data=lrec->tc_dal;
   return 0;
}
```

2.4 Transfer-Protocol

The Gigabit-Link transmits 32-bit words only. The hardware handles proper composition and reports errors or loss of synchronisation to the host. Every transmissione starts with a *special word* (header) and terminates in case of block transfer with a *special word*.

	Bit[31:24] Byte 3	**Bit[23:16] Byte 2**	**Bit[15:8] Byte 1**	**Bit[7:0] Byte 0**	
special word	be: byte enable	sp: space	ctrl: control	SC_PROT Special Char	
AM	Address Modifier Bit 15:0				AM only
ADDR_L	Address A31-0				
ADDR_H	Address A63-32				64-bit only
BC or DATA_L	byte count / datum D31-0				
DATA_H	datum D63-32				64-bit only
DATA	next datum				block transfer only
...					
special word			*CTL*: **END**	SC_PROT	

local protocol format (little endian)

The individual 32-bit words are transmitted over the Gigabit link starting with byte 0 (exception hatched field, see next chapter). A protocol starts with the special character SC_PROT. This byte is transferred as first byte of the *special word*. The content is 0x1C (SC_PROT, K28.0) and is transmitted by the transport layer as special character

The driver can be freed from the burden of single transfer byte swapping in case of a Big Endian remote station (VME e.g.) by enabling swapping with the BIGENDIAN bit in the control register. The PLX bridge chip has swapping capability on a DMA transfer. The BIGENDIAN bit in the control register has no impact on register data.

2.4.1 Little/Big Endian

The local data representation is Little Endian. All address data (AM as well), register contents and byte count are transmitted in Big Endian format during a block transfer. The corresponding entries are hatched in the table. Swapping is handled in hardware without driver intervention. Seen Big Endian system the protocol has the format shown below.

	Bit[31:24] phys. Byte 0	**Bit[23:16] Byte 1**	**Bit[15:8] Byte 2**	**Bit[7:0] Byte 3**	
special word	SC_PROT Special Char	*CTL*: control	sp: space	be: byte enable	
AM	Address Modifier Bit 15:0				with AM only
ADDR_L	Address A31-0				
ADDR_H	Address A63-32				64-bit only
BC/DATA_H	datum D31-0/byte count				
DATA_L	datum D63-32				64-bit only
DATA	next datum				Block transfer only
...					
special word	SC_PROT	*CTL*: **END**			

Die Übertragung des Headers und der Registerinhalte erfolgt auf dem Lichtwellenleiter im Big Endian Format wie unten dargestellt ist. Data words are transmitted in the physical order byte by byte. Data words can be swapped however by setting the BIGENDIAN bit in the control register.

2.4.2 64 Bit CPU's

A 64/bit mode is foreseen in the protocol for the connection of a 64-bit remote station. This mode si not completely implemented however for the time being. The transfer of 64 bit data words has to be signaled in "byte enable" by bits 7:4. The address has to be 8-byte aligned in this case.

2.4.3 Protocol Header, special word

Every protocol sequence starts with a *special word*. The *special word* consists of the special character SC_PROT followed by three more bytes.

Bit	Byte	Bit	Kommentar
7-0 000000FF		SC_PROT 0x1C always	
9-8 00000300	*CTL* (*Control*)	00 **REQ** 01 **END** 10 **CON** 11 **ECON**	request, protocol start end on block transfer positive confirmation error Confirmation
10 00000400		**WR** Write Request	0: Read 1: Write
11 00000800		**AM** use Address Modifier	an Address Modifier can be present in the request protocol only
12 00001000		**A64** 64-bit addresse	for request protocol only
13 00002000		**BT** block transfer	the address is followed by the byte count (1 word) in a read request
14 00004000		**FIFO** constant address	with **BT** (block transfer) only
15 00008000		**EOT** DMA end	may be set in a protocol **END** *special word* only (page **33** and **36**).
21-16 003F0000	**SP** un- modified return expected	0: register 1: direct bus access 2: DMA-Kanal 0 3-63: unused	remote space
23-22 00C00000		0: normal transfer 1: buffer pipe 2: DMA0 pipe 3: unused	local space, used for *pipelined read* mode only
31-24 FF000000	**BE** (request) **EC** (confirm)	01 Byte 0 02 Byte 1 04 Byte 2 08 Byte 3 F0 Byte 7-4	Byte enable is ignored for a register transfer, the value should be 0x0F however, as a 32-bit word is transferred always Two data words (64-bit) will be expected with enable of byte 7-4 non zero The byte holds the error confirmation (**ECON**) in the error case

For a Big Endian Station the special word has the format:

Bit	Byte	Bit	Function
7-0 000000FF	**BE** (request) **EC** (confirm)	01 Byte 0 02 Byte 1 04 Byte 2 08 Byte 3 F0 Byte 7-4	Byte enable is ignored on a register transfer. A value of 0x0F is recommended however, as a 32-bit word is transferred. Two data words (64-bit) are expected if byte enbable of byte 7-4 is non zero. The byte holds the error code in an error confirmation (**ECON**)
13-8 00003F00	**SP** space, un-modified return expected	0: Register 1: direct bus access 2: DMA-chanel 0 3-63: frei	Remote space, the first three are valid for use with the SIS1100 PCI card
15-14 0000C000		0: normaler Transfer 1: Buffer pipe 2: DMA0 pipe 3: unused	Local space, used for *pipelined read* mode
17-16 00030000	*CTL* (*Control*)	00 **REQ** 01 **END** 10 **CON** 11 **ECON**	Request, protokol start End of block transfer Confirmation, positive acknowledge Error Confirmation
18 00040000		**WR** Write request	0: Read 1: Write
19 00080000		**AM** Address Modifier present	A VME address modifier can be part of a request protocol only.
20 00100000		**A64** 64-bit address	For request protocol only
21 00200000		**BT** block transfer	The address is followed by the byte count (one word) in a read request protocol.
22 00400000		**FIFO** constant address	only in conjunction with **BT** (block transfer)
23 00800000		**EOT** DMA end	Can be set in protocol **END** *special word* only (see Page **33** and **36**).
31-24 FF000000		Space for SC_PROT 0x1C always	

Assignment of the Byte Enable Bits

The byte enable bits refer to the physical byte address as shown in the table below for little and big Endian CPUs.

Adresse End Bits	Byte Enable	Little Endian	Big Endian 32 Bit CPU
..00	01	7:0	31:24
..01	02	15:8	23:16
..10	04	23:16	15:8
..11	08	31:24	7:0

Adress End Bits	Byte Enable	Little Endian	Big Endian 64 Bit CPU
..000	01	7:0	63:56
..001	02	15:8	55:48
..010	04	23:16	47:40
..011	08	31:24	39:32
..100	10	39:32	31:24
..101	20	47:40	23:16
..110	40	55:48	15:8
..111	80	63:56	7:0

The 4 byte Enable bits are swapped if the BIGENDIAN bit is set in the control register.

2.4.4 Synchronous and pipeline mode

All direct driver read accesses are synchronous by nature. I.e., wait states will delay the read cycle until receiption of the confirmation. Wit a short fibre (10 m e.g.) a cycle time of **approx. 1µs** plus remote side access time has to be considered. In the case of a CAMAC remote station (SIS5100) the remote side access time will be in the order of 2µs e.g.. Direct write access is handled in a pipelined manner (post write). The write access is ended by the driver as soon as the protocol request has been send. The error is stored in the error register **prot_error** if it empty upon receiption of a confirmation with error. Hence the first write error is known. Wenn der Treiber dieses Fehlerregister ausliest, wird der Lesezyklus solange verzögert, bis das letzte Confirmation empfangen ist (until REQ/**CON p_balance** up/down counter reaches 0). Der driver will be aware of an error state, but will not know during which specific cycle.

A pipelined mode is foreseen for the read cycle (*pipelined read*). Data are stored in system memory directly in this case. Pipeline requests (i.e. addresses to read from) are passed under program control via the temporary transfer registers **t_hdr** to **t_adl** or via DMA from system memory.

A special mode is foreseen for DAQ mode, receiption of readout data respectively. Data are transmitted through DMA channel 0 in this case. Receiption of **EOT** ends the transfer and closes the DMA channel normally wird normalerweise der Transfer beendet und der DMA Kanal geschlossen. The DMA channel stays open in the second method and the byte counter is written to systen memory with every **EOT** and the byte counter is cleared. Physical address and length of the *EOT byte count buffers* are passed by the driver. The address is incremented upon every **EOT** and the length is decremetned by 4. The driver has to monitor the entries into the counter buffer and to digest the data

from the DMA buffer. It can use a variety of options from the descriptor list, that is executed by the PLX bridge chip from system memory. It can use **EOT** interrupt generation also.

The bytes ***BE***(byte enable) snd ***SP***(local/remote space) are expected back without modification in an error free confirmation. nverändert zurück erwartet. In the *CTL*(control) byte **REQ** has to be replaced with **CON** (Bit 1:0). Byte **REQ** is replaced with **ECON** in the *CTL* byte in case of an error confirmation. Byte ***SP*** is inchanged and the the ***BE*** byte holds the error code.

2.5 Transparent Test Mode

This mode was implemented for test purposes. A simple loopback test can be done with a send/receive loopback fibre.

Transparent mode is activated through the TRANSPARENT bit of the control register. As long as the FIFO_FULL bit of the status register is not set, data can be written as "special word" or "normal data" via the **TP_SPECIAL** or **TP_DATA** registers to the output in FIFO . A timer is started when the output FIFO is almost full (see **BUS_TOUT** in status register).

Input FIFO data are flagged with the *TP_SPECIAL* and *TP_DATA* registerts of the status register and can be read from the **TP_SPECIAL TP_DATA** respective registers. Read access will result in a timeout if no data word is pending (**BUS_TOUT** in status register). To guarantee the proper order of special- and data- words, no further data words are read as soon as a special word is pending. The following code example shows a simple loopback that retransmits special words and data words in the same order of their receiption.

```
gigal_regs->cr =CR_TRANPARENT;
while (1) {
   while (gigal_regs->sr &SR_TP_DATA) {
      gigal_regs->tp_data=gigal_regs->tp_data;
   }

   while ((gigal_regs->sr &(SR_TP_DATA|SR_TP_SPECIAL)) == SR_TP_SPECIAL) {
      gigal_regs->tp_special =gigal_regs->tp_special;
   }

}
```

2.6 Local transfer protocols

There are different ways to start a protocol. It starts with a request (**REQ**) in the special header word always and can be initiated by one of the listed actions:

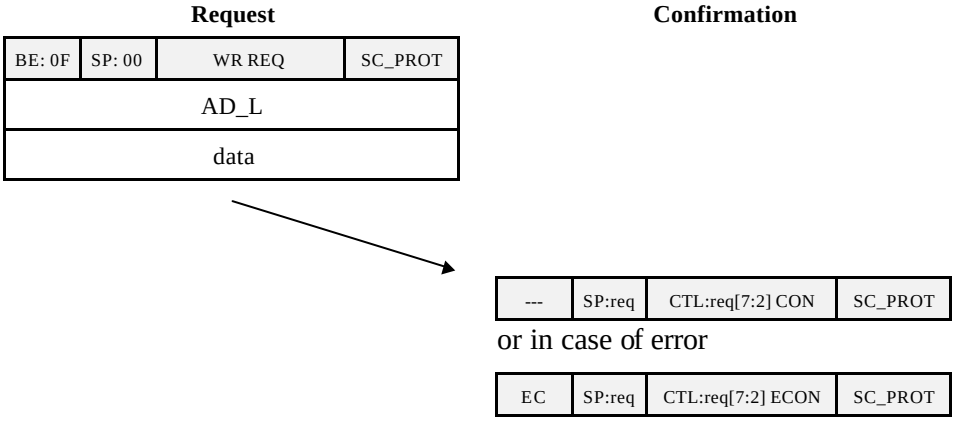
1. direct driver access to the second half of **SPACE0** (remote register)
2. direct driver access to **SPACE1** (direct remote side bus access)
3. Write to the temporary transfer registers
4. DMA1 write request

For the following protocol sequences the writing be:req und sp:req indicates, that the same return value as in the request is expected as response. The sequence CTL:req[7:2] indicates, that the same occupation for bits 7:2 as in the request is expected as response (refer to Fehler! Verweisquelle konnte nicht gefunden werden.).

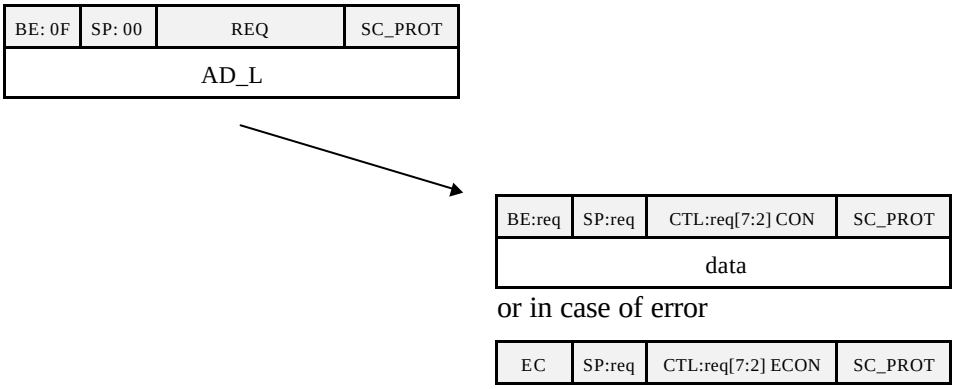
2.6.1 Remote Register

The *Remote Register* set is mapped to the second half of **SPACE0**. It can be accessed through a temporary protocol (with the *Local Registers* **t_hdr** to **t_dah**) also.

Write remote register



Read remote register



The error code is stored into the **prot_error** register (see **2.3.8**). A 0 flags a successful operation. Upon read of **prot_error** the register is restored to 0. The **p_balance** register has to be set to 0 after a *LE_TO* error also. The errors listed below can occur.

Code	Function
LE_SYNCH	no synchronisation, transmission impossible (TX_SYNCH not set
LE_RESOURCE	**XOFF** set in status register, i.e. reception blocked
LE_TO	No protokol confirmation received
RE_TO	Protocoll incomplete
RE_PROT	Format error (possibly wrong address)

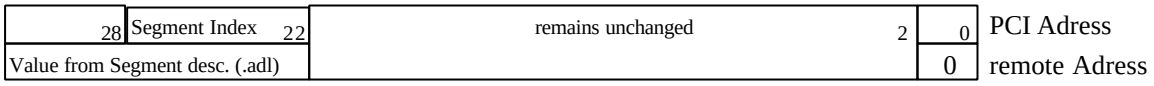
64 Bit remote CPU

This protocol has to be interpreted by a remote 64/bit CPU also.

2.6.2 Direct bus access

The direct bus access protocol is initiated by a **SPACE1** access. **SPACE1** space has a preliminary space of 4MB, a later extension to 256MB is under consideration however. erweitert. The area is subdivided into 64 segments , every segment has ist own descriptor. The 64 descriptor blocks are contained in the register set of descr[0] to descr[63]. A descriptor consists of ***hdr***, ***am***, ***adl*** und ***adh***. The bits **AM** and **A64** of the descriptor byte ***hdr.ctl*** are evaluated. The ***hdr.sp*** byte is used without modification. The actual PCI byte enable signals are entered into bits 3-0 of the byte enable protocol header byte.

The address will be composed as follows for 256MB extension. Bits 31:22 are taken from the descriptor word ***adl***, address bits 21:2 are taken from the actual address. Bits 27-22 of the current address select the descriptor.



Write remote Bus

Request

BE:0x	SP:0r	[AM][A64]WR REQ	SC_PROT
AM (if CTL:AM)			
AD_L			
AD_H (if CTL:A64)			
DA_L			

Confirmation

---	SP:req	CTL:req[7:2] CON	SC_PROT

or in case of error

EC	SP:req	CTL:req[7:2] ECON	SC_PROT

Read remote Bus

BE:0x	SP:0r	[AM][A64]REQ	SC_PROT
AM (wenn CTL:AM)			
AD_L			
AD_H (if CTL:A64)			

BE:0x	SP:req	CTL:req[7:2] CON	SC_PROT
DA_L			

or in case of error

EC	SP:req	CTL:req[7:2] ECON	SC_PROT

The same holds for the error code as for a register access in the previous chapter.

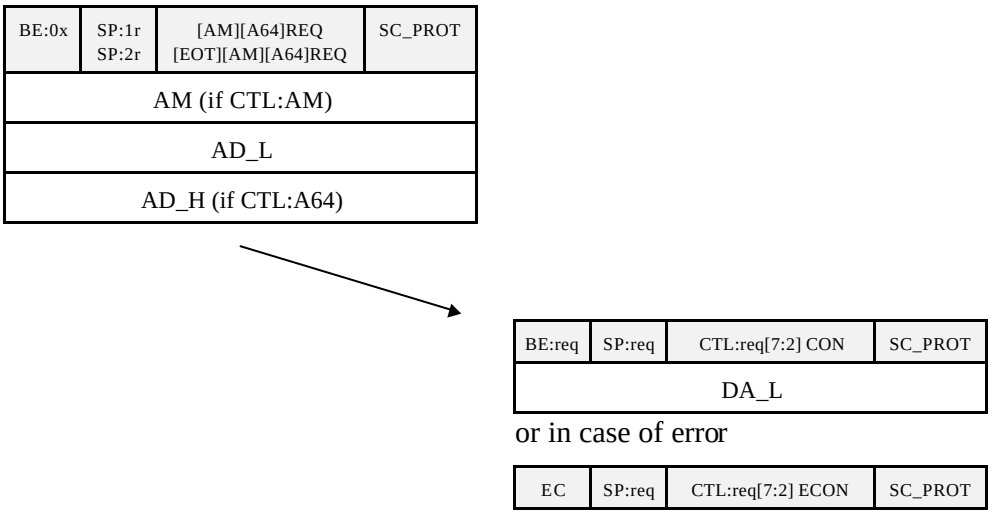
2.6.2.1 Pipelined Read Access

The *pipelined read access* protocol differs in the *local space* of the ***SP*** header bytes. Bits 7:6 are used for local space and assigned as described below:

[7:6]

01 (1) data words are transferred to system memory directly. The physical address and the length of the memory segment has to be stored in the registers ***rd_pipe_buf*** and ***rd_pipe_blen***. If the legth, that is decremented after every confirmation by 4, reaches 0, the error condition *LE_RESOURCE* will be set and nor further data are read from the input FIFO . Status bits **S_XOFF** and **NO_PREAD_BUF** of the status register are set.

pipelined read remote Bus



Error codes in an error confirmation will be stored in register **prot_error** with bit-9 being set. On read the E_DLOCK error will be returned until the balance counter reaches 0 or an error has occured. If the *LE_RESOURCE* error condition is recognized, the driver has the possibility to read the pending data (provide buffer).

Note: It makes no sense to check the end condition by polling the error register, as dedlock errors will result. The right approach is to connect the status bit **prot_end** to an interrupt routine (refer to **Direct PCI BUS** also).

pipelined read Request with temporary Protocol

Request protocols can be send through **t_hdr** to **t_adh** registers or with DMA1 as described in the following section. The registers have to be preset as desribed below:

t_hdr preset with following bytes

- **.be** corresponding to the desired access mode (0x03 or 0xC0 for VME Read 16 bit e.g.).
- **.sp** 0x40|*remspace* for local space system memory. *remspace* depends on the remote station and will be 1 for the SIS1100/3100 combination normally
- **.ctl** [**AM**][**A64**].
- **.sc** used as send protocol indication (refer to **2.6.4** page **34**)

t_am, t_adh if required

t_adl if **t_hdr.*sc*** is set to 0x02, the protocol will be send as soon as this address is written to. A sequence of addresses can be read from by consecutive writes to the register e.g.

pipelined read Request mit DMA1

Another option to send a *pipelined read* request is to use the DMA1 channel. A data buffer has to be set up as described below. Two words are required for the individual read request.

BIT		Function
5-0		Remote Space
6		use AM (Address Modifier)
7		unused
15-8		BE, byte enable bits
31-16		Address modifier
31-0		Remote address

The first word contains the partial headers and the address modifier (if bit-6 set), the second word holds the remote address.

Registers ***d_hdr*** and so on, have to preset as desribed below:

d_hdr preset with following bytes

- **.sc** not used
- **.ctl** [**A64**], bit **WR** has to be cleared as it signals a normal block transfer, [**AM**] is ignored in this context (in DMA buffer).
- **.sp** not used (in DMA buffer).
- **.be** not used (in DMA buffer).

d_am,d_adl not used (in DMA buffer).

d_adh if required

d_bc same length as DMA buffer.

Data pairs from the DMA buffer are transmitted as request protocols consecutively. The *pipelined read* identifier 1 is used in local space. An **EOT** is inserted into the request protocol when the ***d_bc*** register reaches zero, this is not mandatory however.

The local address has no meaning during a DMA cycle with the exception of bit 31 (due to **2.2**). The transfer end should be handled by an interrupt routine for the status bit **prot_end.** Polling of register **prot_error** will result in a preliminary deadlock error.

64 Bit remote CPU

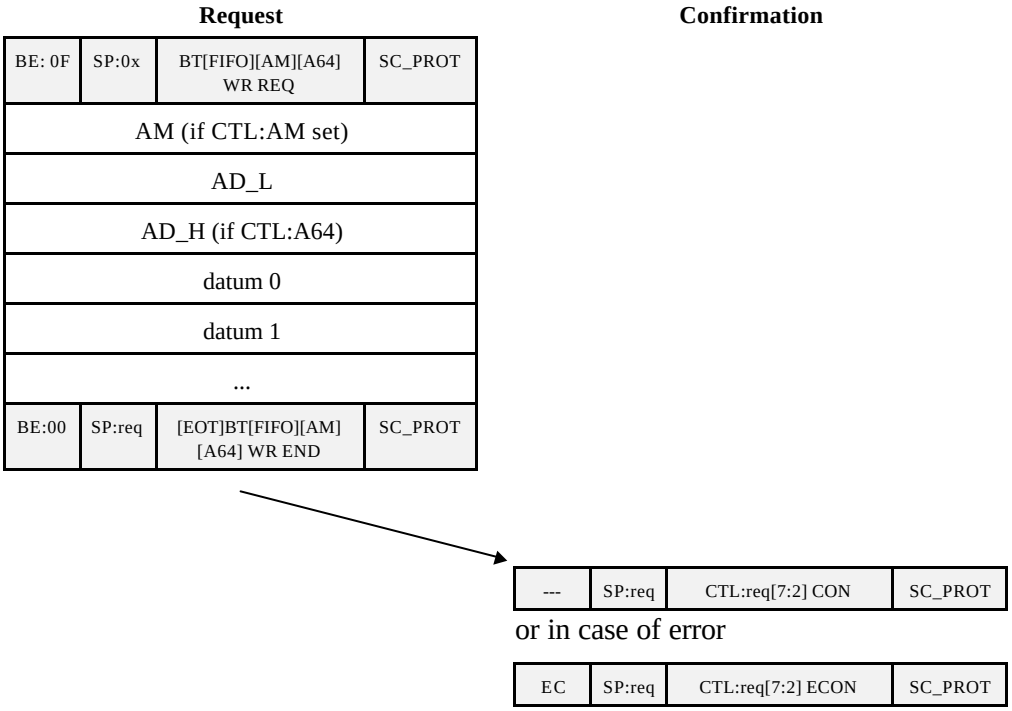
The same as for a normal direct bus access applies.

2.6.3 DMA1 Block Transfer

DMA channel 1, which has no demand mode, is used for write block transfer. In principle the DMA0 channel could be used for this case also. A DMA transfer ist protocol descriptor was assigned, which is represented in the register set under ***d_hdr*** to ***d_adh***. The **WR** bit of byte ***d_hdr.ctl*** has to be set to distinguish between block transfer read and *pipelined read*.

The bit **BT** is set in the header byte *CTL* of the protocol, no byte counter is transmitted. Transmission takes place in the PCI bus induced bursts. As the length of the individual bursts is not know, protocol blocks are terminated with an **END** *special word*. The last block holds the **EOT** code in the **END** *special word*. Bits **AM, A64** and **FIFO** (**WR** has to be at 1) of the descriptor byte ***d_hdr.ctl*** are evaluated and byte ***d_hdr.sp[5:0]*** is used without modification. Byte "byte enable" is set to 0x0F. Bit 31 is taken from the descriptor word ***d_adl,*** the address bits 30:2 are taken from the current address (refer to **2.2** page **4**).

Write Block



The PLX bridge chip has no information on the end of a DMA transfer. Hence the driver has to store the byte count (multiple of 4), in the local register ***d_bc*** to allow for **EOT** generation. The register is decremented by 4 with every DMA word. The **EOT** bit is set as soon as the value reaches 0. It has to be taken into account, that a block tranfer may/will be split into smaller blocks due to the PCI burst nature. The individual blocks are terminated with an end word. The data words are transferred without gaps. A 0 start value for ***d_bc*** will not result in an error.

Read access to the error register **prot_error** before end of DMA transfer will interrupt data ransfer until occurence of a timeout.

Data are transferred transparent (i.e. in little Endian format , the driver can configure the PLX bridge chip for swapping to big endian format however.

2.6.4 Temporary protocol

The user can send a protocol by writing to *Local Registersa* also. Registers **t_hdr** to **t_dah** are used for this purposeb. They are also used to send *pipelined read* protocols and there are other possibilities to use them. The value that is written to **t_hdr.*sc*** defines at what point the protocol will be transmitted.

0x00 no protocol will be send

0x01 protocol will be send upon writing of **t_hdr**.

0x02 protocol will be send upon writing of **t_adl**.

0x04 protocol will be send upon writing of **t_dal**.

The error register **prot_error** has to be checked afterwards

2.6.4.1 Normal Read/Write

A normal Read/Write is executed if bit **BT** in byte **t_hdr.*ctl*** is not set (block transfer) and bits 7:6 in byte **t_hdr.*sp*** are not set also. The protocol will be transmitted if the corresponding register is set up as described earlier. The confirmation has to be checked by reading **prot_error.** The result will be available in **tc_hdr** and **tc_dal/tc_dah** unless an error occured.

2.6.4.2 Pipelined Read

A *pipelined read* is used if bits 7:6 in byte **t_hdr.*sp*** are 1 and bits **BT** and **WR** in byte **t_hdr.*ctl*** are not set. The exact procedure has been desribed earlier (refer to Page **30**).

2.6.4.3 Block Read Request

A block read request is possible with local space 2 (DMA0) only. The temporary protocol registers have to be set up as follows.

t_hdr as word composed by following bytes

- ***.sc*** refer to **2.6.4**.
- ***.ctl*** [**AM**][**A64**][**FIFO**], **BT**
- ***.sp*** 0x80|*remspace* for DMA0 as local space. *remspace*.
- ***.be*** 0x0F normally

t_am, t_adh if required

t_adl Address on remote side

t_dal Byte Counter

The last register to be written to will depend on the value of **t_hdr*.sc***.

Note:

Attention on **prot_error reads,** a deadlock situation may occur. The driver has to check, that all data words were received (afterr **EOT** interrupt).

Read Block

Request

BE: xx	SP:1x 2x	BT[FIFO][AM][A64] REQ	SC_PROT
AM (if CTL:AM set)			
AD_L			
AD_H (if CTL:A64)			
DA_L(byte count)			

Confirmation

BE:req	SP:req	CTL:req[7:2] CON	SC_PROT
data			
...			
---	SP:req	CTL:req[7:2] [EOT]END	SC_PROT

or in case of error

EC	SP:req	CTL:req[7:2] [EOT]ECON	SC_PROT

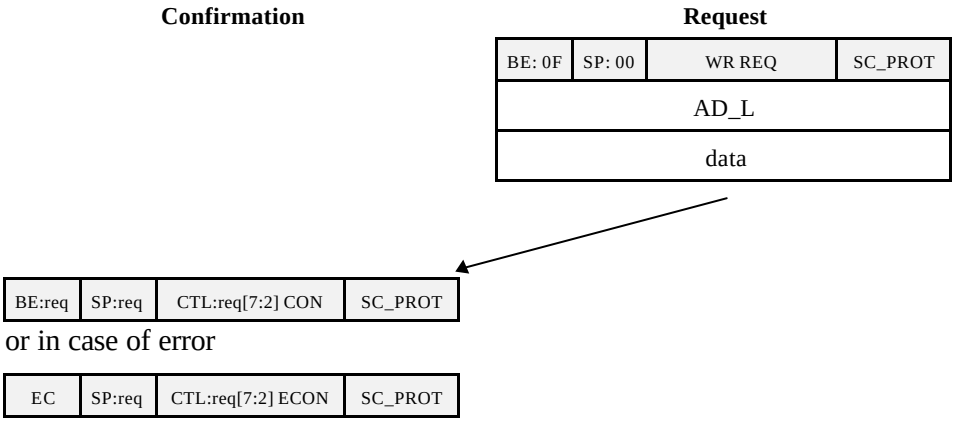
Remote transfer types

This section gives an overview on the protocols, which can be send as request by a remote stationd. Errors which are send from the remote side as response to a request will set the PROT_ERR bit in the status register, which is used as information to the driver only.

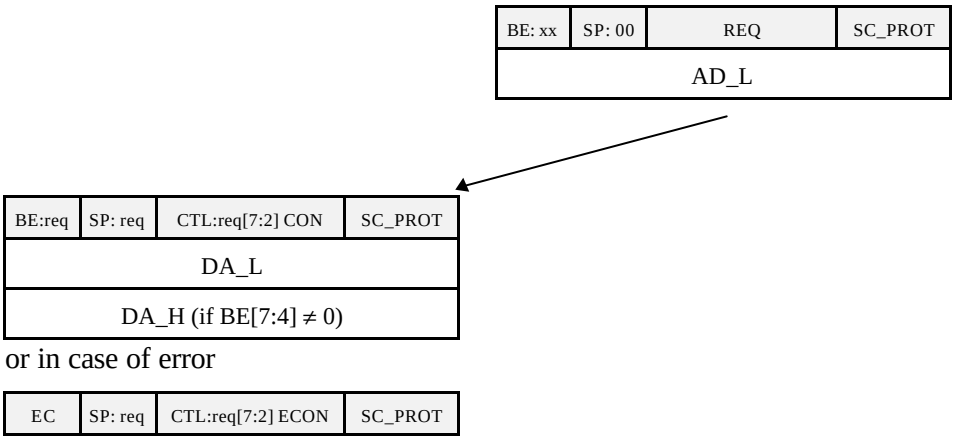
2.6.5 Register Access

In principle the remote station can read and write all registers, the implemented read/write bits will be affected/read only however. The register address space is repeated from address 0x800 on. Registers which are not implemented will be read as 0.

Write register



Read register



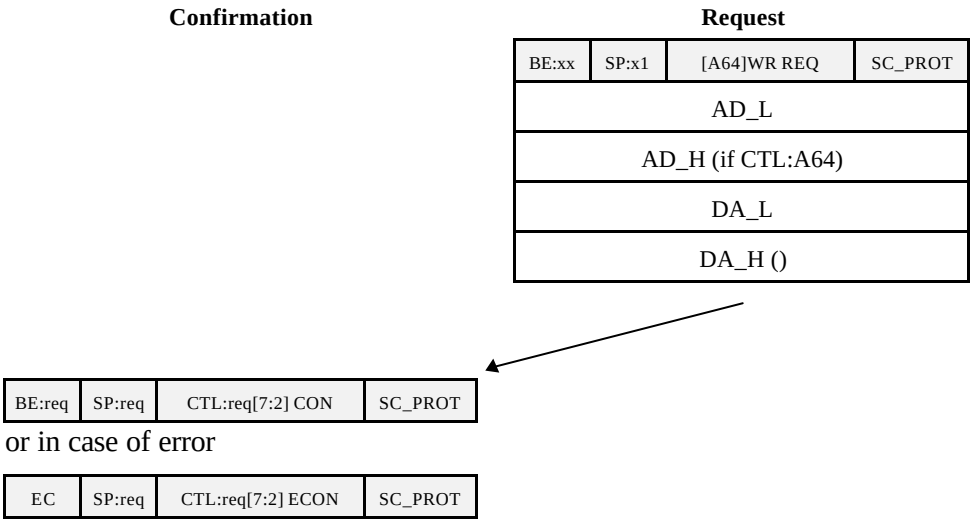
64 Bit remote CPU

Read request only is supported in 64-bit mode, as one can imagine situations in which a 64-bit interface has no information on whether the CPU will want to read the low or the high word.

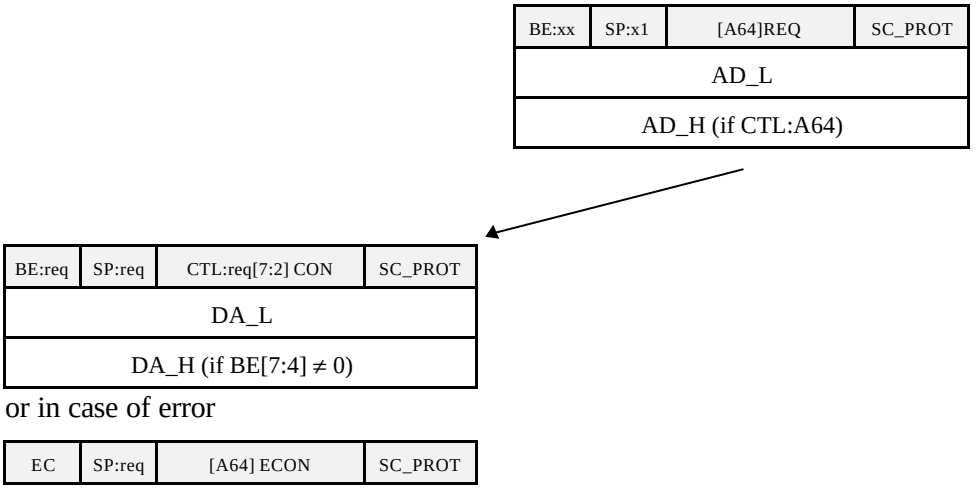
2.6.6 Direct bus access

AM and **A64** and the corresponding values are ingored in the protocol, the remote space has to be 1.

Write Bus



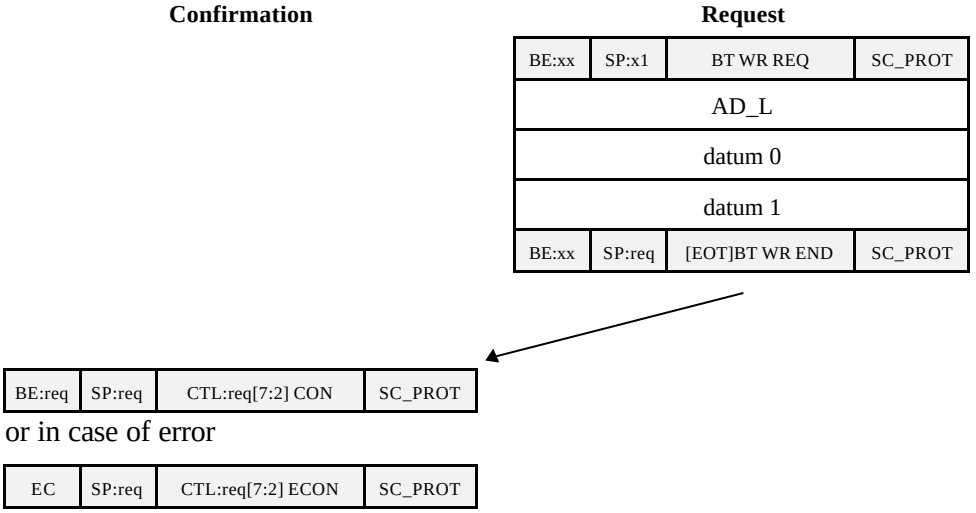
Read Bus



2.6.7 Block Write Transfer

During a block transfer data are read from system memory directly. The remote space has to be 1. The Byte enable code is ignored. Complete words are read only, the format is defined as shown below.

Write Block



The end identification **END** is not mandatory. An arbitrary new request protocol can be used as end. A confirmation protocol will be send upon end identification **END** only. Information will be passed to the driver on the local side upon reception of a **EOT** end identification.

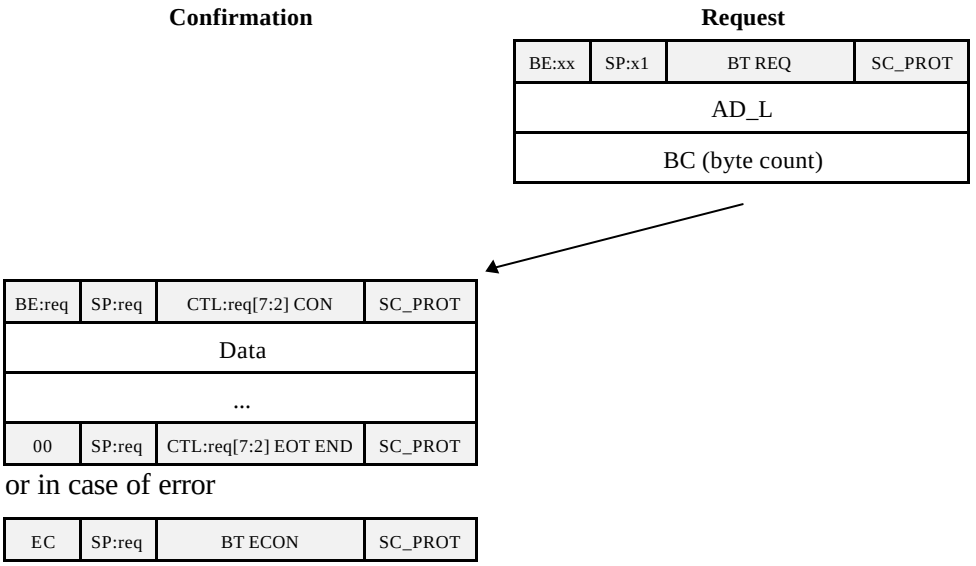
64 Bit remote CPU

64 Bit mode (**A64**) not allowed.

2.6.8 Read Block

During a block transfer data are read from system memory directly. The remote space has to be 1. The Byte enable code is ignored. Complete words are read only, the format is defined as shown below.

Read Block



Data will be read from system memory beginning with address AD_L. AD_L has to be a valid physical address. The byte counter BC has to be 4-byte aligned (i.e. bit 1:0 are ignored)

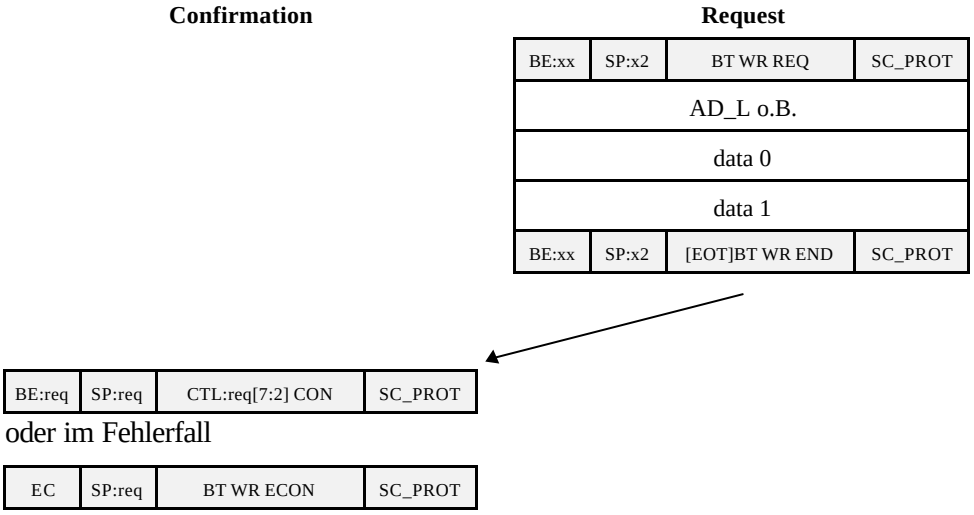
64 Bit remote CPU

64 Bit mode (**A64**) not allowed.

2.6.9 DMA0 Block Transfer (DAQ Mode)

Der DMA 0 Kanal wird immer für demanded Transfer von remote zum Systemspeicher programmiert. Er wartet sozusagen auf ein Write Request Protokoll mit Space DMA0. Die Adresse in diesem Protokoll ist ohne Bedeutung.

Write DMA0



Die Endekennung **END** ist nicht unbedingt erforderlich. Als Protokoll Abschluß kann auch ein beliebiges neues Request Protokoll dienen. Die Endekennung **END** kann aber auch zu einem späteren Zeitpunkt als einzelnes Wort gesendet werden. Das Confirmation Protokoll wird aber nur auf die Endekennung gesendet. Wenn zwischen dem letzten Datenwort und dem **END** mit **EOT** eine zeitliche Lücke ist, wird noch ein Dummy Wort übertragen, um dem DMA Kanal ein EOT Signal zu übergeben. Im Byte Counter wird das Dummy Wort aber nicht gezählt.

Die Daten werden kontinuierlich übertragen. Sobald ein **EOT** in der Endekennung erkannt wird, wird der DMA Transfer beendet. Ein Treiber kann sich dies durch einen Interrupt mitteilen lassen. Da der PLX Chip keine Angabe über die aktuelle Länge gibt, ist ein Bytecount Register implementiert, welches sich im Registersatz unter ***d0_bc*** befindet. Dieses Register wird mit jedem Wort um 4 inkrementiert. Es wird durch einen beliebigen Schreibzugriff auf 0 gesetzt.

Für das kontinuierliche Sammeln von Daten ist ein besonderer Mode vorgesehen. Wenn der Treiber in dem Register ***d0_bc_buf***/***d0_bc_blen*** die Adresse und Länge eines *EOT bytecount buffers* übergibt, wird bei jedem **EOT** der aktuelle Byte Counter ***d0_bc*** in den Systemspeicher geschrieben und anschließend der Byte Counter auf null gesetzt. Die ***d0_bc_buf*** Adresse wird nach jeder Byte Count Übertragung um 4 inkrementiert und die Länge entsprechend dekrementiert bis sie den Wert 0 erreicht. In diesem Fall, wie auch in dem Fall, daß der DMA0 Kanal gestoppt ist, werden keine weiteren Daten aus dem Input FIFO übertragen. Die remote Seite kann dann keine weiteren Protokolle übertragen und der lokalen Seite wird ein Fehler gemeldet, wenn sie versucht ein Protokoll zu übertragen. Für die Bereinigung dieser Situation ist der Treiber verantwortlich (siehe auch **Direct PCI BUS** Seite **6**).

64 Bit remote CPU

Ein 64 Bit Mode (**A64**) ist hier nicht erlaubt.

3 Appendix

3.1 Protocol overview

All protocols are shown in Big Endian Format

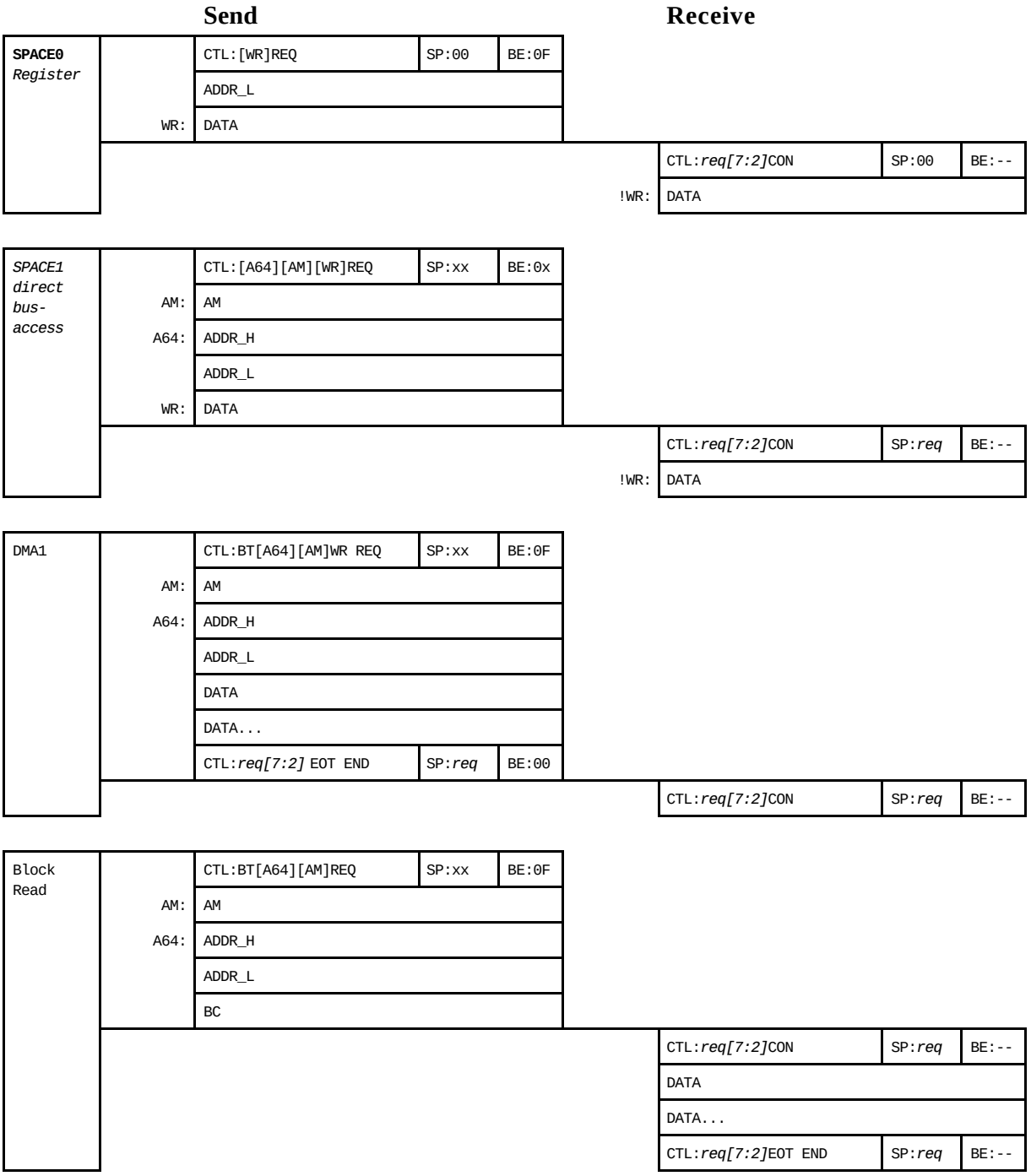
`--` not relevant

`[]` may be set

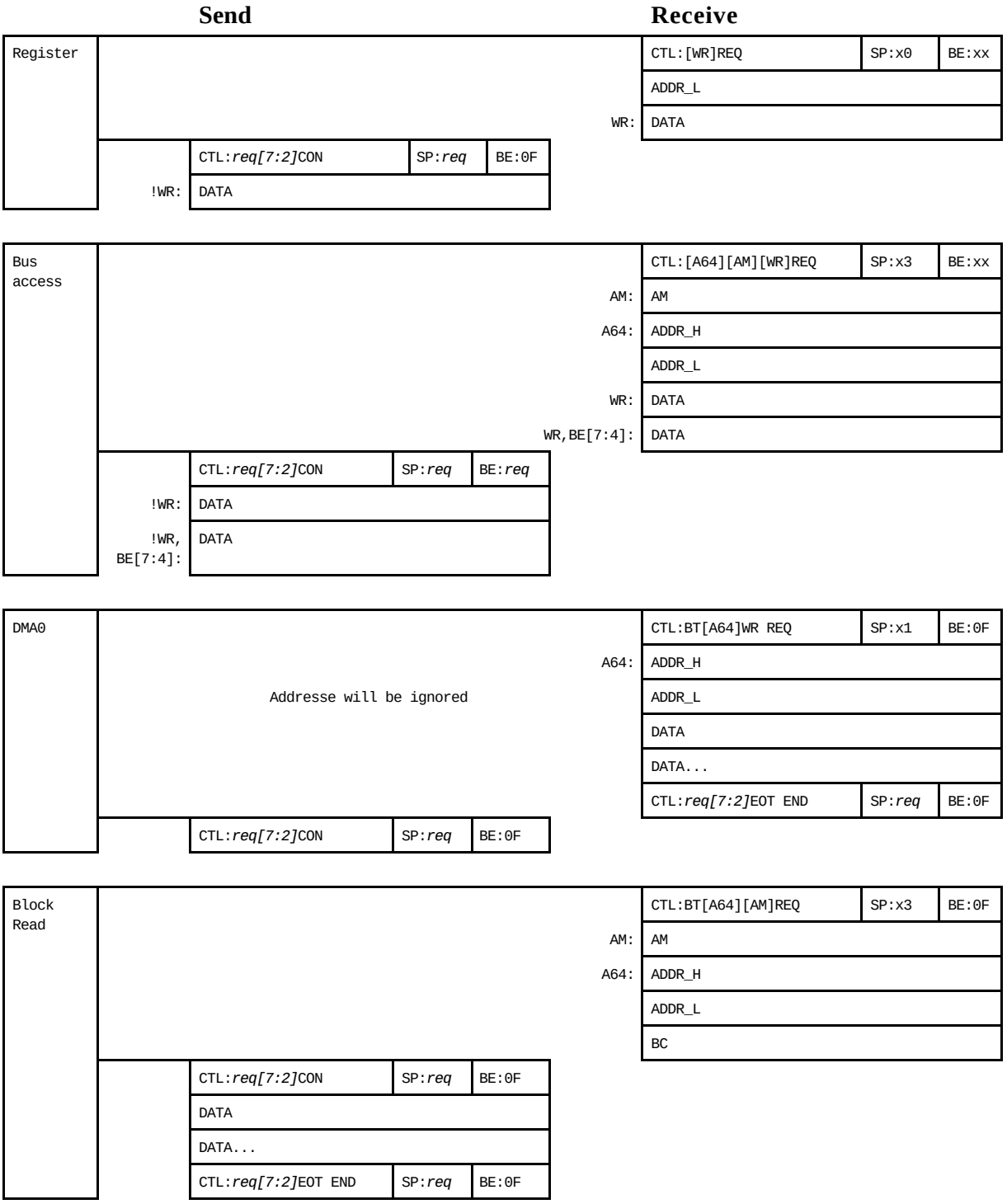
`req` same value as in REQ protocol

`!WR` **WR** bit not set (i.e. RD)

`xx` arbitrary value (hexadecimal)



Protocols from the remote station



3.1.1 C Definitions

Der lokale PLX Treiber arbeitet im wesentlichen mit 4 Pointern, um lokale Register, Register auf der remote Seite und Speicher auf der remote Seite direkt zu adressieren.

1. a pointer to the PLX device registers (refer to the PLX manual)
2. a pointer to the local registers (first half of **SPACE0**)
3. a pointer to the remote registers (second half of **SPACE0**, offset 0x0800)
4. a pointer to **SPACE1** for direct bus access to the remote side

```
typedef struct regspace {
      u_int32     ident;
      u_int32     sr;
      u_int32     cr;
      u_int32     semaphore;
      u_int32     doorbell;
      u_int32     res0[3];
      u_int32     mailbox[8];
      u_int32     res1[16];

      u_int32     t_hdr;
      u_int32     t_am;
      u_int32     t_adl;
      u_int32     t_adh;
      u_int32     t_dal;
      u_int32     t_dah;
      u_int32     res2;

      u_int32     tc_hdr;
      u_int32     tc_dal;
      u_int32     tc_dah;     // not yet implemented

      u_int32     p_balance;
      u_int32     prot_error;

      u_int32     d0_bc;
      u_int32     d0_bc_buf;
      u_int32     d0_bc_blen;

      u_int32     d_hdr;
      u_int32     d_am;
      u_int32     d_adl;
      u_int32     d_adh;
      u_int32     d_bc;
      u_int32     res4[2];

      u_int32     rd_pipe_buf;
      u_int32     rd_pipe_blen;
      u_int32     res5[2];

      u_int32     tp_special;
      u_int32     tp_data;
      u_int32     opt_csr;
      u_int32     jtag_csr;
      u_int32     res6[2];

      u_int32     mailext[192];

      struct {
            u_int32     hdr;
            u_int32     am;
            u_int32     adl;
            u_int32     adh;
      } sp1_descr[64];
} REGSPACE;
```

3.1.2 Index