VME - ISER8

Intelligent VMEbus Board with 10 Serial Interfaces

Hardware Manual

N O T E

The information in this document has been carefully checked and is believed to be entirely reliable. **esd** makes no warranty of any kind with regard to the material in this document, and assumes no responsibility for any errors that may appear in this document. **esd** reserves the right to make changes without notice to this, or any of its products, to improve reliability, performance or design.

esd assumes no responsibility for the use of any circuitry other than circuitry which is part of a product of **esd** gmbh.

esd does not convey to the purchaser of the product described herein any license under the patent rights of **esd** gmbh nor the rights of others.

esd electronic system design gmbh
Vahrenwalder Str. 205
30165 Hannover
Germany

Phone: +49-511-37298-0
FAX: +49-511-37298-68
E-mail: info@esd-electronics.com
Internet: www.esd-electronics.com

USA / Canada
7667 W. Sample Road
Suite 127
Coral Springs, FL 33065
USA

Phone: +1-800-504-9856
FAX: +1-800-288-8235
E-mail: sales@esd-electronics.com

Manual File: i:\texte\doku\manuals\vme\iser8\ISER824H.ENG

Described PCB version	ISER82

Changes in the chapters

The changes in the user's manual listed below affect changes in the **firmware**, as well as changes in the **description** of the facts only.

Chapter	Changes Versus Rev. 2.1
-	Separate hardware and software parts of the manual.
1.2	Technical data revised.
1.4.1	Default base address corrected to $xx800000.
1.4.3	Device no. of SRAM corrected.
1.5.2	CPU 68000 / 20 MHz inserted.
1.6.2	Jumper J5 deleted.
1.9.2	Circuit diagram of new RS-485 piggyback with termination resistor array.
2.	CLK-Signals of RS-422 and RS-485 interface of channel 9 corrected.

Further technical changes are subject to change without notice.

User's Manual ISER8

Content **Page**

1. Hardware

1.1 Block Diagram

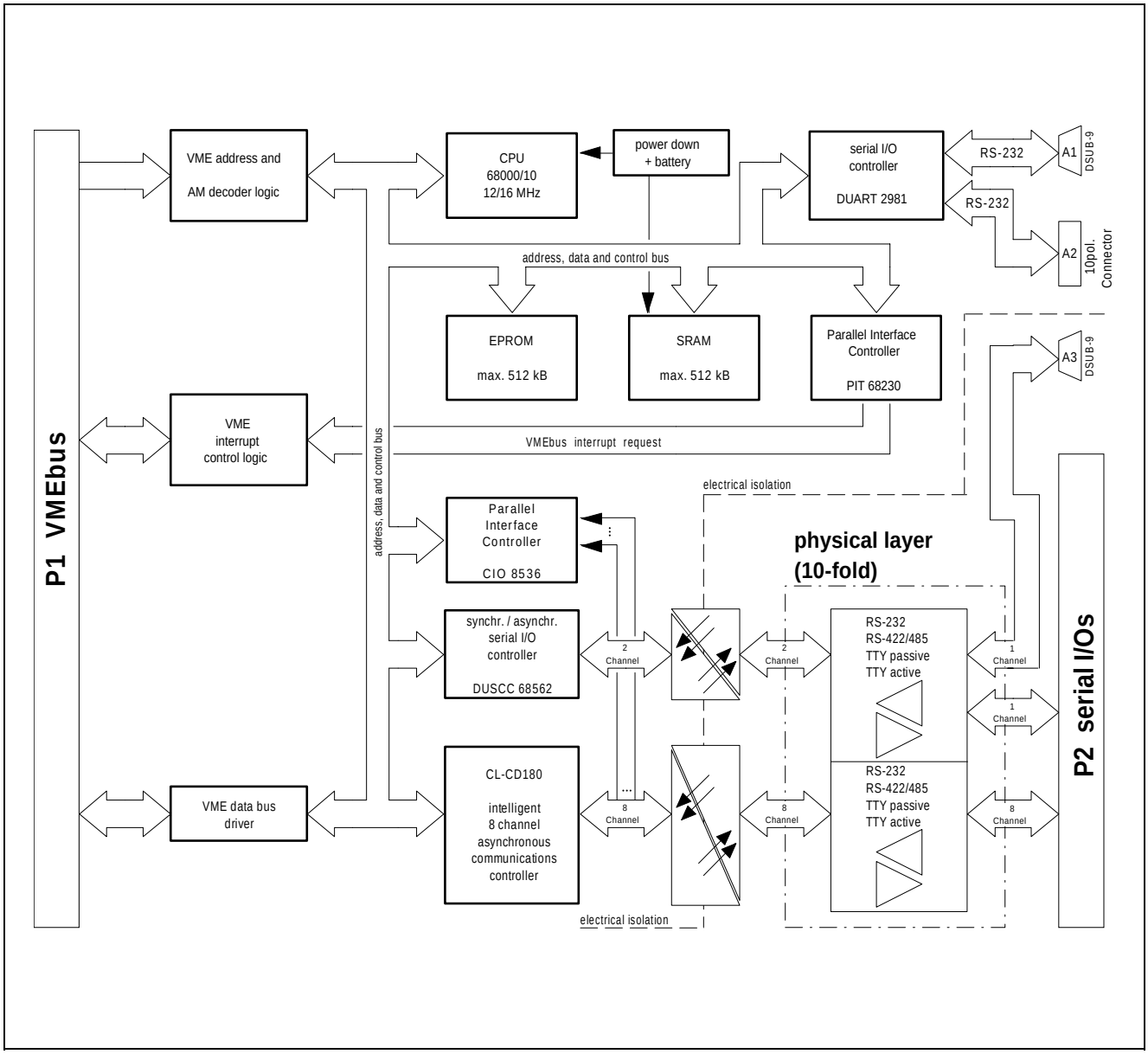


Fig. 1: Block Diagram of the VME-ISER8

1.2 Technical Data

The VMEbus board **VME-ISER8** is an **intelligent** interface board equipped with **10 serial process interfaces** and **two additional** serial ports for programming and service purposes.

The **VME-ISER8** contains a local **MC 68000 CPU** with **20 MHz** and up to **512 kB shared SRAM** for the execution of complex data transfer protocols.

The transfer frame of each serial channel can be separately selected between **5** and **8** bits with a **parity bit as an option** (odd, even, with or without parity) and **one** or **two stop bits**.

The **maximum baud rate** when using all 10 serial channels simultaneously is **38.4 kBaud**. For each channel a **software** handshake (XON/XOFF) or a **hardware handshake** can be selected.

As physical layers for the serial transfer channels for each channel **RS-232, RS-422, RS-485 or TTY current loop** (active or passive) are selectable.

The interfaces can be combined as follows:

1. **RS-232, TTY active, TTY passive**

2. **RS-232, TTY passive, RS-485, RS-422**

The combinations TTY active and RS-485 or TTY active and RS-422 are not possible.

The additional ports for programming and service purposes are always realized as RS-232-interfaces.

Two of the 10 process interfaces can also be operated **synchronously**. For these 2 transfer channels several **bit-oriented protocols** (BOP: HDLC, SDLC, X.25, X.75) and **character-oriented protocols** (COP: BISYNC, DDCMP) with **CRC generation** and **detection** and several **data code formats** (e.g. NRZ, NRZI, FM0, FM1, Manchester) are available.

All serial data lines and handshakes of the 10 process interfaces are **optoisolated** from the VMEbus section by fast **optocouplers**. ±12 V is supplied by an on-board DC/DC converter.

Control and test software are available **in EPROMs as a standard**. The operation ensues **via the VMEbus** or **via a serial terminal**.

1.2.1 VMEbus Process Controller

- 16/32 bits - microprocessor **MC68000 20MHz**
- 128 kbytes buffered SRAM (up to 512 kbytes SRAM)
- 128 kbytes EPROM(up to 256 kbytes EPROM)
- **VMEbus** interface A32/D16 and A24/D16 according to **Rev. C**
- **WATCHDOG** circuit
  (not supported by the local software at the moment)
- 2 serial interfaces **RS232C** / 19200 Baud
- **Interrupts** on the VMEbus
- interrupt controlled bipolar message-interface

1.2.2 Serial Process Interfaces

- **10 asynchronous** serial interfaces, **2** of them even **synchronous** as well:

- physical layers (to be ordered as an option):
  RS-232, RS-422, RS-485, TTY current loop (active or passive)

- baud rate (asynchron):
  up to **38.4 kBaud** (at CL-CD180), up to **115 kBaud** (only at DUSCC 68562) (higher baud rates are possible with DUSCC in synchron mode)

  To avoid data loss the baud rate sum of all active channels (10 Rx-channels + 10 Tx-channels = 20 channels) should not exceed approx. 800 kBaud!

- electrical isolation: by fast optocouplers

1.2.3 Real-time Software

- **Real-time/Multitasking** operating system RTOS-UH

- **firmware**

- **various asynchronous data transfer protocols** as the German protocols **SAE8, TWG, ZM400, SEAB, SINEC, 3964R or Multidrop** are available.

- the **shared RAM interface** makes the implementation of **different operating systems**, e.g. **OS-9, UNIX, PDOS, VxWorks, VRTX32 or RTOS-UH** easy to realise; drivers for **OS-9** are available.

1.2.4 General

VMEbus Interface	IEEE 1014 / C.1
data transfer mode	SADO32 - slave with A32/D16 access SD16 - slave with A24/D16 access
temperature range	0...50 °C
humidity	max. 90%, non-condensing
connector type	P1 - DIN 41612-C96 P2 - DIN 41612-C96 A1 - DSub 9 female A2 - 10 pole connector plug A3 - DSub 9 female
board size	160 mm x 233 mm
VMEbus dimensions	6 U height, 1 slot width
weight	620 g
power supply	VMEbus P1: 5 V ±5% / 2.5 A VMEbus P1: +12 V ±5% / 1.0 A -12 V ±5% / 0.1 A

Table 1: General Technical Data of the VME-ISER8

1.3 Address Covering of the VME-ISER8

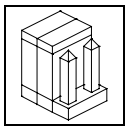
address bits VME address on-board address 31 24 16 8 0	board addr.	component short designation	explanation
eeee.eeee.sss0.1111.1111.1111.1111.1111 eeee.eeee.sss0.1100.0000.0000.0000.0000	0FFFFF 0C0000	EPROM bank	system EPROM range (256 kbytes maximum)
eeee.eeee.sss0.1010.xxxx.xxxx.xxxx.xxxx	0A0000	WATCHDOG	time-out watchdog in MAX695
eeee.eeee.sss0.1000.0111.x0xx.xxxx.xxx1	087001	VFR	**V**MEbus **f**orced **r**eset
eeee.eeee.sss0.1000.0111.x1xx.xxxx.xxx1	087401	VFA	**V**MEbus **f**orced **a**bort
eeee.eeee.sss0.1000.0110.xxxx.xx11.1111 eeee.eeee.sss0.1000.0110.xxxx.xx00.0001	08603F 086001	PIT	**p**arallel **i**nterface/**t**imer 68230
eeee.eeee.sss0.1000.0101.xxxx.xxxx.x111 eeee.eeee.sss0.1000.0101.xxxx.xxxx.x001	085007 085001	CIO	**c**ounter/timer and parallel **I**/**O** unit Z8536
eeee.eeee.sss0.1000.0100.xxxx.xxx1.1111 eeee.eeee.sss0.1000.0100.xxxx.xxx0.0001	08401F 084001	DUART (channel A+B)	**du**al **a**synchronous **r**eceiver/ **t**ransmitter SCN2681
eeee.eeee.sss0.1000.0011.xxxx.xxxx.xxx1	083001	RTC	**r**eal-**t**ime **c**lock
eeee.eeee.sss0.1000.0010.xxxx.xxxx.xxx1	082001	IR vector	VMEbus slave interrupt logic
eeee.eeee.sss0.1000.0001.xx01.1111.1111 eeee.eeee.sss0.1000.0001.xx01.0000.0001	0811FF 081101	OCTART (channel 1...8)	intelligent **oct**al channel **a**synchronous **r**eceiver/ **t**ransmitter CL-CD180
eeee.eeee.sss0.1000.0001.xx00.x111.1111 eeee.eeee.sss0.1000.0001.xx00.x000.0001	08107F 081001	DUSCC (channel 9+10)	**d**ual **u**niversal **s**erial **c**ommuni- cations **c**ontroller SCN 68562
eeee.eeee.sss0.1000.0000.xxxx.xxxx.xxx1	080001	IRFIRE	VMEbus slave interrupt logic
eeee.eeee.sss0.1000.1111.1111.1111.1111 eeee.eeee.sss0.0100.0000.0000.0000.0000	07FFFF 040000	RAM2	SRAM bank 2 (max. address range 256 kbytes)
eeee.eeee.sss0.0001.1111.1111.1111.1111 eeee.eeee.sss0.0001.0000.0000.0000.0000	01FFFF 010000	RAM2	SRAM bank 2 (min. address range 64 kbytes)
eeee.eeee.sss0.0011.1111.1111.1111.1111 eeee.eeee.sss0.0000.0000.0000.0000.0000	03FFFF 000000	RAM1	SRAM bank 1 for operating syst. (max. address range 256 kbytes)

eeee.eeee.sss ....extended address access VMEbus
(A20 - A31 = CARD ADDRESS)

sss .....standard address access VMEbus
(A20 - A23 = CARD ADDRESS)

x .....not evaluated address bits

Table 2: Address Covering of the VME-ISER8



1.4 Jumpers Configuration

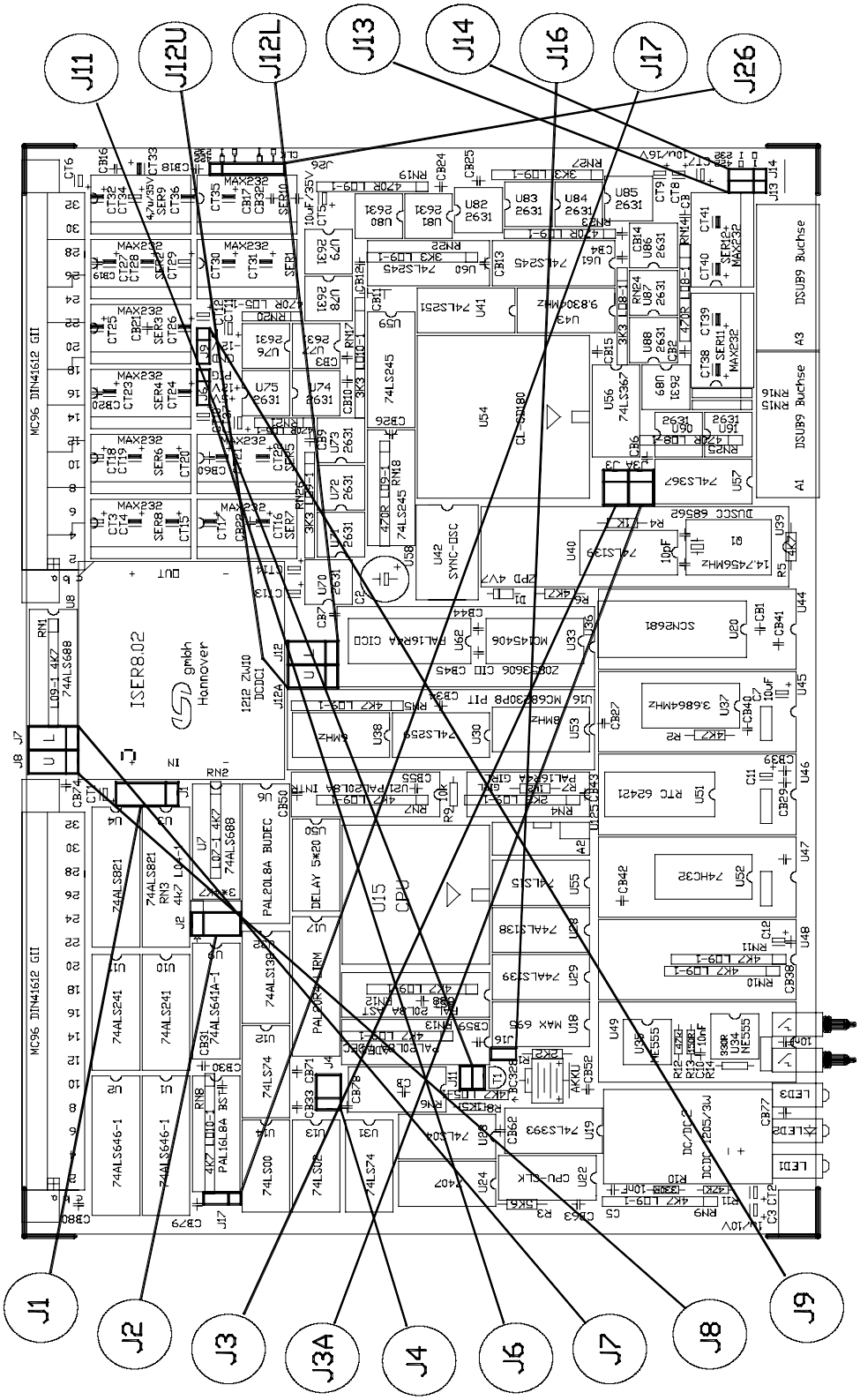


Fig. 2: Position of the Jumpers on the ISER8

1.4.1 Default Setting

The respective default setting is factory-set as follows in table 3.

The position of the jumpers can be obtained from the insertion diagram.
An inserted jumper corresponds to a ’0’(Low) level of a signal.

In the following the jumpers are displayed in a position, as seen by the user, if he has the board lying in front of him with the VMEbus connectors to the rear end.

Default setting of the jumpers J1 to J14:

jumper	function	setting
J1	address modifier AM	AM2 don’t care, i.e. access at supervisory and user mode VME access A24/D16
J2	addresses A20..A23	base address $xx800000
J3	sync clock for serial channel 10	not inserted, i.e. no synchronous operation possible
J3A	sync clock for serial channel 9	not inserted, i.e. no synchronous operation possible
J4	DTACK delay	min. 140 ns at VME access
J6	supply of the piggy-backs	+12V DC/DC decoupled to piggy-backs
J7,J8	addresses A24..A31	not inserted, A24 mode (J1)
J9	supply of the piggy-backs	-12V DC/DC decoupled to piggy-backs
J11	memory size	4 x 32 kbytes SRAM (U44, U45, U47, U48) 2 x 64 kbytes EPROM (U46/U49)
J12U,J12L	configuration	not inserted, i.e. configuration byte=$FF
J13, J14	sync clock channel 10 input/output	not inserted, i.e. no sync. clock

Default setting of the jumpers J16 to J26 :

jumper	function	setting
J16	watchdog MAX695	not inserted (1.2 seconds)
J17	VME-RESET/SYSFAIL	no VME RESET or VME SYSFAIL at CPU reset
J26	sync clock channel 9 input/output	not inserted, i.e. no sync. clock

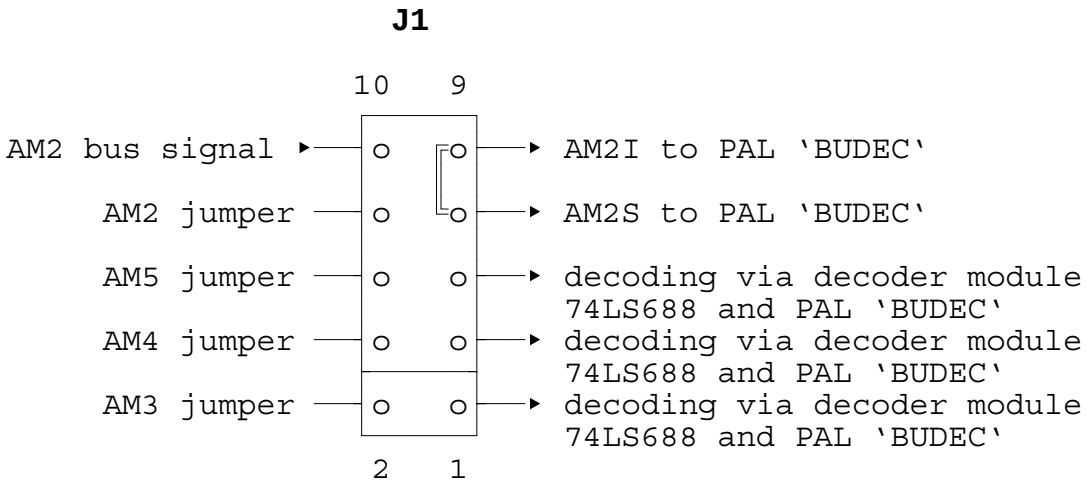
Table 3: Default Setting of the Jumpers

1.4.2 Base Address and AM (J1,J2,J7,J8)

1.4.2.1 The Address Modifiers AM on Jumper J1:

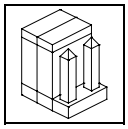
The ISER8 can be operated with the access modes A24/D16 (standard) or A32/D16 (extended).
In the shown default setting access will be done in the A24/D16 mode. AM2 will be ignored, so that the IMC5 can be addressed in the user mode as well as in the supervisory mode.

Default setting: standard supervisory and nonprivileged data access (A24 mode)



The decoding of the AM signals (in PAL 'BUDEC' and 74LS688) on the board does not allow all arbitrary combinations.

The levels of the AM signals, to which the board responds, are listed as follows next page.

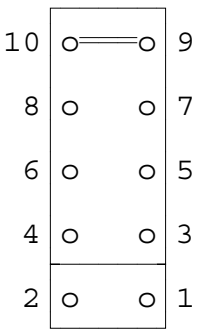


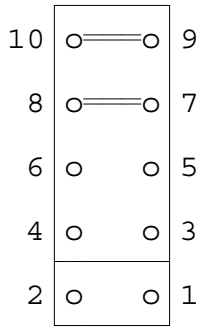
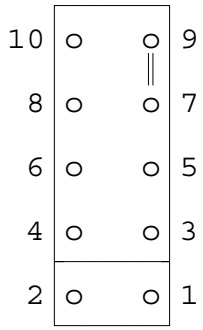
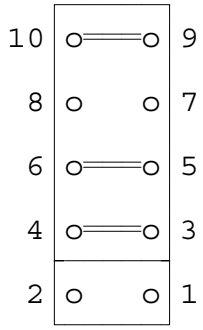
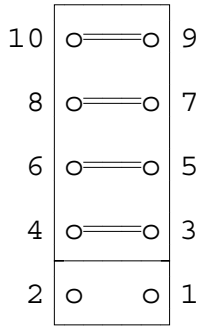
The AM signals of the VMEbus are decoded as follows in the table below:

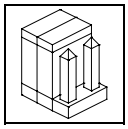
AM Signal	Permissible Levels	Coding via J1
AM0	1	no selection possible
AM1	0	no selection possible
AM2	0 1 0 and 1	jumpers 7-8 and 9-10 inserted jumper 9-10 inserted jumper 7-9 inserted
AM3	0 1	jumper 1-2 inserted jumper 1-2 not inserted
AM4, AM5	0,0 1,1	the board responds only, if AM4 and AM5 have the same level:
AM4 AM5	0 0	jumper 3-4 inserted jumper 5-6 inserted
AM4 AM5	1 1	jumper 3-4 not inserted jumper 5-6 not inserted

Table 4: Permissible Levels of the AM Signals

From the table above efficient jumper combinations and addressing modes result as follows:

J1	Permissible AM Codes							Addressing Mode
	AM5	AM4	AM3	AM2	AM1	AM0	HEX	
	1	1	1	1	0	1	3D	standard supervisory data access (A24)

J1	Permissible AM-Codes							Addressing Mode
	AM5	AM4	AM3	AM2	AM1	AM0	HEX	
	1	1	1	0	0	1	39	standard nonprivileged data access (A24)
	1 1	1 1	1 1	0 1	0 0	1 1	39 3D	standard nonprivileged data access (A24) & standard supervisory data access (A24)
	0	0	1	1	0	1	0D	extended supervisory data access (A32)
	0	0	1	0	0	1	09	extended nonprivileged data access (A32)



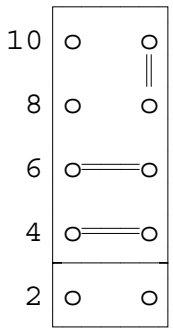
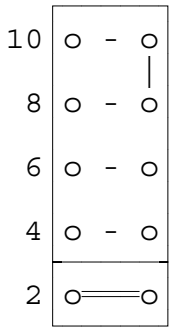
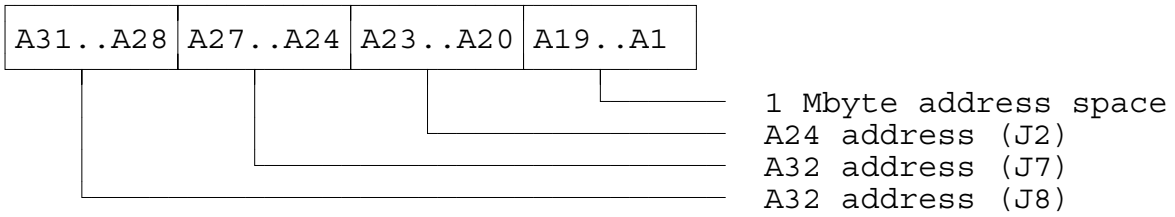
J1	Permissible AM-Codes							Addressing Mode
	AM5	AM4	AM3	AM2	AM1	AM0	HEX	
	0	0	1	0	0	1	09	extended nonprivileged data access (A32) &
	0	0	1	1	0	1	0D	extended supervisory data access (A32)
	a	b	0	x	0	1	-	reserved
	! a = b							note:
	a = 0,1 b = 0,1 x = 0,1							the board responds to combinations with AM3 = 0, though these combinations are reserved according to VMEbus specification ANSI/IEEE SDT1014.
	- possible jumper settings see table 4							

Table 5: Selection of the Addressing Modes (AM Codes) via J1

1.4.2.2 Base Address Setting by Jumpers J2, J7 and J8

The base address of the ISER8 results as follows:

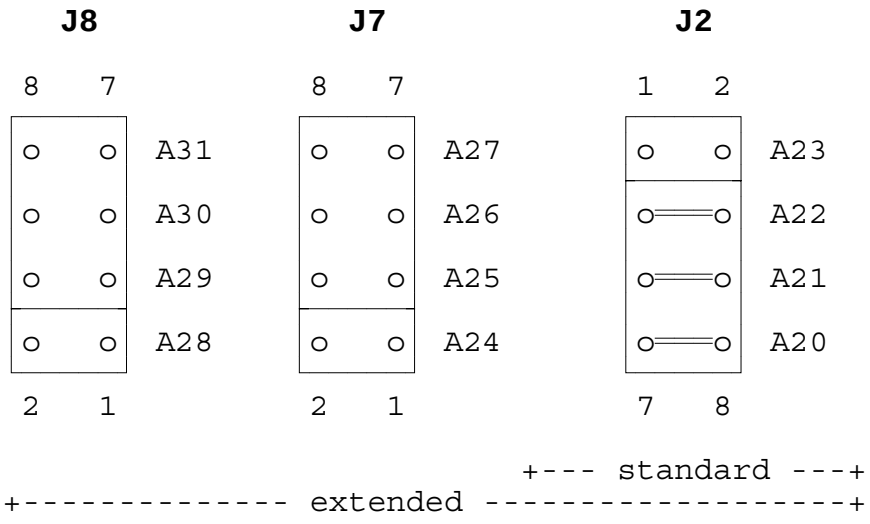


jumperfields J7,J8 : base address for extended addressing codes (A24 - A31)

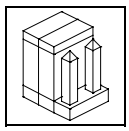
jumperfield J2 : base address for standard addressing codes (A20 - A23)

Note:

If A24 access (standard addressing) is selected by J1, the jumperfields J7 and J8 will not be decoded.
If the A32 access (extended addressing) is selected by J1, the jumperfields J7, J8 <u>and</u> J2 will be decoded (A20-A31)!

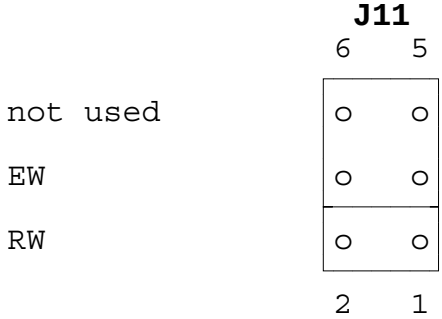


The default setting of the base address of the ISER8 is (A24 mode with J1 inserted) xxC00000 HEX.



1.4.3 Memory Size and DTACK (J4,J11)

Jumperfield J11: fixing of the memory range of RAM banks 1 and 2 and of the EPROM bank



jumper		memory range	
EW	RW	RAM banks	EPROM bank
1	1	bank 1: (U44, U47) $00000-$0FFFF 64 kbytes e.g. 2x 51256 bank 2: (U45, U48) $10000-$1FFFF 64 kbytes e.g. 2x 51256	$C0000-$DFFFF 128 kbytes e.g. 2x 27512
default setting			
1	0	bank 1: (U44, U47) $00000-$3FFFF 256 kbytes e.g. 2x 581000 bank 2: (U45, U48) $40000-$7FFFF 256 kbytes e.g. 2x 581000	$C0000-$DFFFF 128 kbytes e.g. 2x 27512
0	1	bank 1: (U44, U47) $00000-$0FFFF 64 kbytes e.g. 2x 51256 bank 2: (U45, U48) $10000-$1FFFF 64 kbytes e.g. 2x 51256	$C0000-$FFFFF 256 kbytes e.g. 2x 27010
0	0	bank 1: (U44, U47) $00000-$3FFFF 256 kbytes e.g. 2x 581000 bank 2: (U45, U48) $40000-$7FFFF 256 kbytes e.g. 2x 581000	$C0000-$FFFFF 256 kbytes e.g. 2x 27010

0...jumper inserted
1...jumper not inserted

Table 6: Fixing of the local RAM Memory Range by J11

Jumperfield J4 :

Delay of the DTACK signal for VMEbus accesses after chip select of a module.

Note:

The effective delay of the VMEbus DTACK signal after address strobe of the VMEbus cannot be indicated concretly:

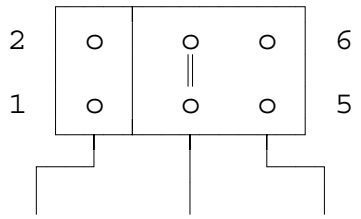
The VMEbus DTACK signal can only be sent, if the VMEbus master has addressed a module on the ISER8. But the VMEbus master can only access to a module, if it has the control of the local data/address bus on the ISER8.

The local bus enable is allocated according to an arbitration routine similar to the one of the VMEbus. That means, if the local CPU has bus control, the VMEbus master has to wait, until the local CPU has finished its access cycle. Only if this is the case, the VMEbus master can start its cycle, can select the module and generate a VMEbus DTACK.

Thus the time delay of the VMEbus DTACK depends not only on the time set by J4. Essential for the length of the time delay are also the clock frequency and the local CPU as well as the operating condition of the local CPU at the moment of the VMEbus request.

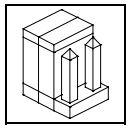
Since these factors depend on the equipment of the board, respectively on chance, in the following only the time between chip select of the memory or of the peripheric modules and the DTACK is indicated:

J4



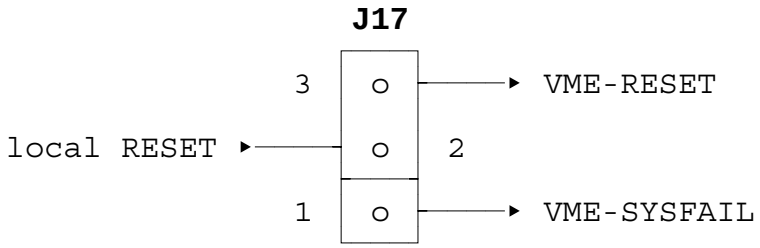
DTACK active about 110ns 140ns 170ns after chip select

The DTACK delay for accesses of the local CPU is not variable.



1.4.4 RESET, Interrupts, Clocks (J3,J16,J17)

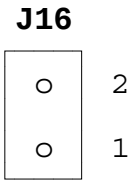
Jumperfield J17 : Connection of the local RESET with the VME RESET, or with the VME SYSFAIL respectively



Standard setting: no jumper inserted

jumper J16 : Setting of the time-out time of the watchdog MAX 695

Jumper not inserted: RESET after 1.2 sec
Jumper inserted: RESET after 100 msec



Standard setting: no jumper inserted

1.4.5 Configuration Field (J12U, J12L)

With the jumpers J12U and J12L, 2 x 4 bits for application dependant configurations are available. 3 of them are reserved for the system section (RTOS-operating system and drivers), 5 more are assigned for the motion controller section.

The signals are fed to port B of the PIT and are continuously readable at address **$86017** (as a byte access).

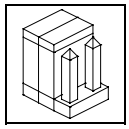
Jumper J12U

				data bit	function	not ins.	inserted
8	o	o	7	D7	reserved	-	-
6	o	o	5	D6	reserved	-	-
4	o	o	3	D5	reserved	-	-
2	o	o	1	D4	free	-	-

Jumper J12L

				data bit	function	not ins.	inserted
8	o	o	7	D3	free	-	-
6	o	o	5	D2	free	-	-
4	o	o	3	D1	free	-	-
2	o	o	1	D0	free	-	-

Table 7: Jumpers J12L and J12U Assignment to the Bits of the Configuration Byte



1.4.6 Jumpers of the 10 Serial Channels
(J3, J3A, J6, J9, J13, J14, J26)

Jumper J6 and J9: supply of the piggy-backs with +12V/-12V or with +5V via a DC/DC converter

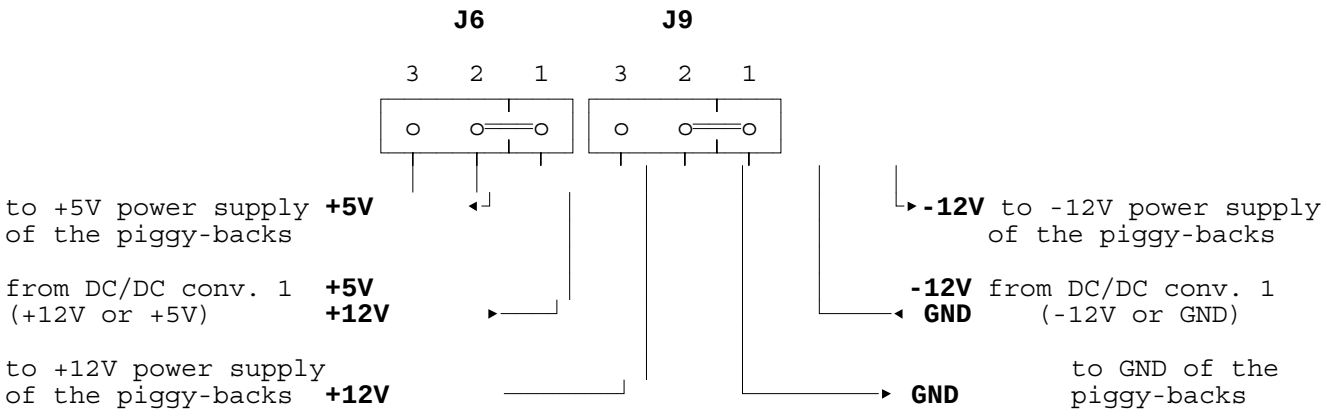
The insertion of the piggy-backs for the serial channels 1 to 10 may be performed with modules, which can require different supply voltages.
For this reason the ISER8 can be operated with different DC/DC converters.

Normally a ±12V DC/DC converter for the piggy-backs (DC/DC1) and a +5V DC/DC converter for the optocouplers (DC/DC2) are inserted.

The DC/DC converter for the optocouplers is always realized as a +5V type.

The DC/DC converter for the piggy-backs is realized as +5V or as ±12V type depending on the selected interfaces.

If as DC/DC converter 1 a +5V type is inserted, it replaces the DC/DC converter 2.
In this case the DC/DC converter 2 is not inserted!

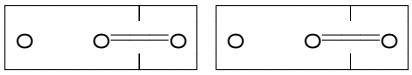
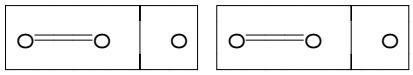


Attention !

If these jumpers are inserted wrong, the piggy-backs or the DC/DC converters might be destroyed!

Please note the table following on the next page for the positions of J6 and J9!

The jumpers J6 and J9 have to be inserted according to the desired piggy-back insertion, also depending on the DC/DC converters:

piggy-backs	RS-232	X	X
	RS-485	not permissible	X
	RS-422	not permissible	X
	TTY passive	X	X
	TTY active	X	not permissible
DC converters	DC/DC1	+12V/-12V type	+5V type
	DC/DC2	+5V type	not inserted
jumpers J6 and J9		**J6** **J9** 3 2 1  3 2 1 **default setting**	**J6** **J9** 3 2 1  3 2 1

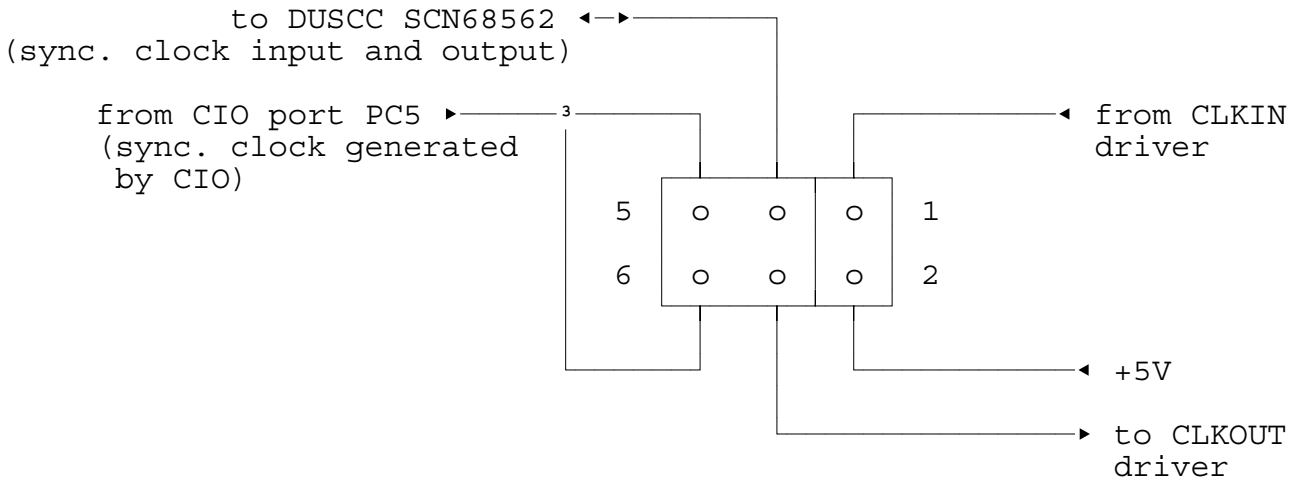
Only these two combinations of the positions of J6 and J9 may be <u>used</u>!
The combinations TTY active and RS-485 or TTY active and RS-422 are not possible.

Table 8: Assignment of the jumpers J6 and J8 to the inserted piggy-backs and DC/DC converters

Jumperfield J3 and J3A: Selection and assignment of the synchronous clock signals for the serial channels 9 and 10

The jumpers J3 and J3A accomplish the same function.
J3 is inserted for channel 9 and is inserted J3A for channel 10.

Covering of the jumper fields:



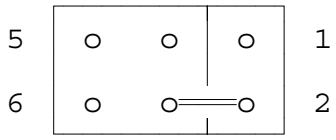
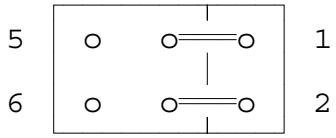
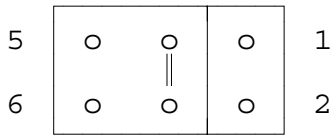
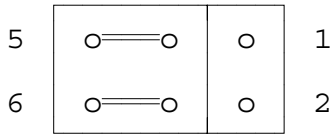
selection of the sync. clock	**J3 or J3A**
no synchronous operation **default setting**	
sync. clock feeded externally	
sync. clock generated internally by DUSCC SCN68562	
sync. clock generated internally by CIO Z8536	

Table 9: Selection of the Sync. Clock Generator via J3 and J3A

Jumperfield J26: Fixing of the sync. clock direction at the different serial transfer modes for channel 9

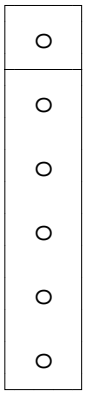
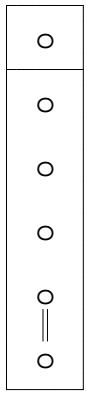
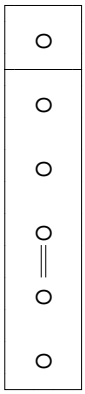
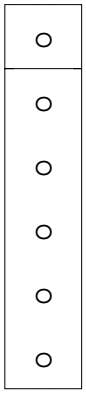
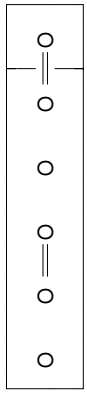
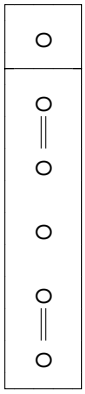
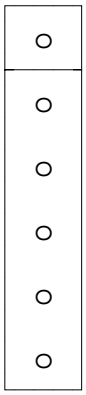
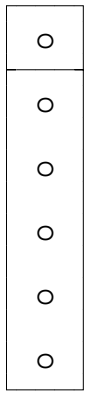
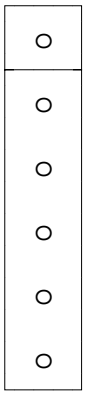
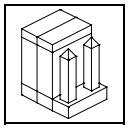
transfer mode	asynchronous: **(default setting)** jumper J26 not inserted!	synchronous	
		sync. clock generated on the ISER8	sync. clock generated externally
RS-232			
RS-485 RS-422			
20mA (TTY) no sync. operation			

Table 10: Fixing of the Sync. Clock Direction for Channel 9



Jumperfields J13, J14: Fixing of the sync. clock direction at the different serial transfer modes for channel 10

transfer mode	asynchronous: **(default setting)** jumpers J13, J14 never inserted!	synchronous	
		sync. clock generated auf ISER8	sync. clock generated externally
RS-232	**J13 J14** 3 o o 3 2 o o 3 1 o o 1	**J13 J14** 3 o o 3 2 o o 2 ‖ 1 o o 1	**J13 J14** 3 o o 3 ‖ 2 o o 2 1 o o 1
RS-485 RS-422	**J13 J14** 3 o o 3 2 o o 3 1 o o 1	**J13 J14** 3 o o 3 ‖ ‖ 2 o o 2 1 o o 1	**J13 J14** 3 o o 3 2 o o 2 ‖ ‖ 1 o o 1
20mA (TTY) no sync. operation	**J13 J14** 3 o o 3 2 o o 3 1 o o 1	**J13 J14** 3 o o 3 2 o o 2 1 o o 1	**J13 J14** 3 o o 3 2 o o 2 1 o o 1

Table 11: Fixing of the Sync. Clock Direction for Channel 10

1.5 Insertion of the SRAM and EPROM Modules

1.5.1 Fixing of the Memory Size and of the Memory Range

The selection of the memory size is performed by the jumperfield J11. By this also the location of the memory range is defined.

The descriptions of jumper J11 and of jumper J4, by which the DTACK delay of the VMEbus accesses is defined, have already been stated in the section 'memory size and DTACK'.

1.5.2 Assignment of the EPROM and SRAM Access Times to the Local CPU Clock Frequency

EPROM or SRAM accesses of the local CPU will enable the local DTACK signal without a considerable delay. By this the processing speed of the board is far higher.

The combination of the memory access time and of the inserted CPU must be selected as follows for an error-free data transfer:

CPU/clock frequency	EPROM max. access time	SRAM max. access time
68000/8MHz 68010/8MHz	200ns	150ns
68000/10MHz 68010/10MHz	150ns	100ns
68000/12MHz 68010/12MHz	150ns	100ns
68000/16MHz	120ns	50ns (80ns) 1*)
68000/20MHz	120ns	50ns (80ns) 1*)

*1) If faster PALs are used for the address decoding, 20 to 30nsec can be saved

Table 12: Assignment of the Memory Module Access Times to the Clock Frequency of the inserted CPU

1.6 Interrupt Processing

1.6.1 Covering of the Local Processor Interrupts

All interrupts listed in the table below are enabled after a RESET, i.e. if an interrupt condition is applying, the corresponding interrupt will be generated!

The interrupt levels of the local CPU of the ISER8 are covered as follows:

IR level	module	type	driver
7	ABORT	auto vector	RTOS
6	DUART-IRQ	auto vector	RTOS
5	RECEIVER (OCTART)	user vector	firmware
4.1	DUSCC(synchronous)	user vector	firmware
4.2	CIO-IRQ	user vector	-
3.1	TRANSMIT (OCTART)	user vector	firmware
3.2	PIT-Timer-IRQ	user vector	-
2	GIRL-IRQ (mailbox)	user vector	firmware
1	STATUS (OCTART)	user vector	firmware

Table 13: Covering of the Local Interrupt Levels

If you need any further informations on the local interrupts, please contact **esd**.

1.6.2 VMEbus Slave Interrupt Logic

The VMEbus slave interrupt logic offers the opportunity, to generate a VMEbus interrupt by the ISER8 (PIT). Moreover, from the VMEbus a local interrupt can be generated via the PAL GIRL.

1.6.2.1 Generation of a VMEbus Interrupt via the PIT 68230

Besides the timer interrupt of the PIT (TIRQ*), the PIT can also generate an interrupt at port PC5IRQ. This interrupt releases VME IRQ and is additionally fed to the CIO pin PB3.
The interrupt is released via the PIT inputs H1...H4.
On a changing edge of these input signals the PIT generates an interrupt and outputs a corresponding vector in the interrupt routine. The further interrupt handling of the PIT is described in detail in the data sheet of the PIT 68230.

The interrupt level on the VMEbus is defined via the PIT ports PC0, PC1 and PC2:

PIT port			VMEbus interrupt level
PC2	PC1	PC0	
0	0	0	no IRQ
0	0	1	IRQ1
0	1	0	IRQ2
0	1	1	IRQ3
1	0	0	IRQ4
1	0	1	IRQ5
1	1	0	IRQ6
1	1	1	IRQ7

Table 14: VMEbus Interrupt Level defined by PIT Port C0, C1 and C2

The access (starting the interrupt) to the signals H1...H4 ensues by application of the respective addresses:

signal	function
IRFIRE*	'chip select'
A1	A1=1 --▸ Hx=1 A1=0 --▸ Hx=0 (x = 1...4)
A2,A3,A4	selection of H1...H4

Table 15: Access to the PIT Inputs H1...H4 via Address Bits

The inputs H1...H4 are set to ’0’ at a RESET. In the following table those addresses are listed, which set the corresponding H signals to ‘1‘ or to ‘0‘.

address	interrupt signal	level
$080011	H4	==▸ 0
$080013	H4	==▸ 1
$080015	H3	==▸ 0
$080017	H3	==▸ 1
$080019	H2	==▸ 0
$08001B	H2	==▸ 1
$08001D	H1	==▸ 0
$08001F	H1	==▸ 1

Table 16: Setting of the PIT Inputs H1...H4 via Address Bits

1.6.2.2 Generating a Local Interrupt via the VMEbus (PAL 'GIRL')

Four different interrupts can be generated via the PAL GIRL, started by the application of the corresponding interrupts. These interrupt signals (F0...F3) can apply 4 x 16 different interrupt vectors to the local data bus, together with the previously programmed data bits. The PAL generates a local interrupt of level 4 of the CPU while 'firing' one of the interrupt inputs.

Following, firstly the programming of the interrupt vectors and then the starting of the interrupts will be described in the form of tables.

1.6.2.3 Programming and Reading-back of the Interrupt Vectors

Address of the interrupt vector: **$082001**

Write:
The data bits D0 to D3 can be freely programmed.

Read:
The data bits D0 and D1 indicate the *actually* applying interrupt with the highest priority coded at (F0...F3).
The data bits D2 and D3 are always read as '1'.
The bits D4 to D7 reflect the previously programmed bits D0 to D3.

Sequence description:

1. Condition of the register after the falling edge of the select signal IRV* - **write**

data bits

7	6	5	4	3	2	1	0
D3 stored	D2 stored	D1 stored	D0 stored	**D3 input**	**D2 input**	**D1 input**	**D0 input**
don't care	don't care	don't care	don't care	**input**	**input**	**input**	**input**

2. Condition of the register after the rising edge of the select signal IRV* - **write**

data bits

7	6	5	4	3	2	1	0
D3 stored	D2 stored	D1 stored	D0 stored	**D3 input**	**D2 input**	**D1 input**	**D0 input**

shift D0...D3 ==▸ D4...D7 and store

3. Condition of the register when reading in the interrupt acknowledge cycle
   - **read**

data bits

7	6	5	4	3	2	1	0
D3 stored	D2 stored	D1 stored	D0 stored	1	1	Fcy transp	Fcx transp

programmed data bits for interrupt vector

always to '1'

actual interrupt of the highest priority:

Fcy	Fcx	interrupt
0	0	F0
0	1	F1
1	0	F2
1	1	F3

1.6.2.4 Generating a Local Interrupt via the PAL GIRL

The local interrupt of GIRL PAL is generated similarly to the VMEbus interrupt of the PIT. The four interrupt signals (F0...F3) are set to low (‘0‘) or to high (‘1‘) by application of addresses. In the opposite to the PIT the GIRL PAL does, however, not generate an IRQ at the change of an edge of the interrupt signal, but at the high level of the signals.

The access to the signals F0...F3 ensues by application of the corresponding addresses:

signal	function
IRFIRE*	‘chip select‘
A1	A1=1 --▸ Fx=1 A1=0 --▸ Fx=0 (x = 0...3)
A2,A3,A4	selection of F0...F3

Table 17: Access to the GIRL PAL Inputs F0...F3 via Address Bits

The inputs F0...F3 are set to ‘0‘ at a RESET. In the following table the addresses are listed, which set the corresponding H signals to ‘1‘ or to ‘0‘.

address	interrupt signal	level
$080001	F0 **reserved**	==▸ 0
$080003	F0 **reserved**	==▸ 1
$080005	F1 **reserved**	==▸ 0
$080007	F1 **reserved**	==▸ 1
$080009	F2	==▸ 0
$08000B	F2	==▸ 1
$08000D	F3	==▸ 0
$08000F	F3	==▸ 1

Table 18: Setting of the GIRL PAL Inputs F0...F3 via Address Bits

1.7 Watchdog Circuit

The applied MAX695 has tasks as follows:

1. generation of the RESET signal at POWER-UP

2. generation of the RESET signal at lacking watchdog trigger. The watchdog must previously be activated, if a RESET is desired (see ‘RESET, Interrupts, Clocks‘). Locally the function is always effective with inserted firmware.

3. generation of a RESET signals at remaining under the operating voltage tolerance

If the watchdog is triggered once, it must be retriggered within a certain time. This time is about 1.6 seconds with J13 not inserted and about 0.1 seconds with J13 inserted.
The watchdog is retriggered by writing to or reading from the WATCHDOG register $A0000 (see mapping: ‘Address covering of the ISER8‘).

1.8 External Generation of a local RESET (VFR) or ABORT (VFA)

The user has the possibility to generate a hardware ABORT or a hardware RESET on the ISER8 via the VMEbus.
With this the RESET line of the local CPU is pulled to ‘low‘ (0 V) via the hardware. Analogously to this at an ABORT by hardware an interrupt is generated at level 7 of the local CPU.

The RESET and the ABORT line are activated by writing of a ‘dummy byte‘ to the following addresses (relatively to the base address of the ISER8):

RESET: $87001

ABORT: $87401

1.9 The Serial Interfaces

1.9.1 Overview

The ISER8 contains 10 serial process interfaces. 8 of them are handled by the OCTART CL-CD180 and 2 of them are handled by the DUSCC SCN68562.

(Two additional ports for programming and service purposes are handled by the DUART 2681 and are designed as RS-232 interfaces.)

The maximum baud rate when using all 10 serial channels simultaneously is 38.4 kBaud. For each channel software handshake (XON/XOFF) or hardware handshake can be selected.

Each of the 10 channels can be operated optionally as RS-232, RS-422, RS-485 or TTY current loop (active or passive). The different transfer modes can be realized by means of the RS-232 driver MAX232 or by piggy-backs.

The interfaces can be combined as follows:

1. **RS-232, TTY active, TTY passive**

2. **RS-232, TTY passive, RS-485, RS-422**

(The combinations TTY active and RS-485 or TTY active and RS-422 respectively are not possible.)

In the first case the power supply of the drivers is +12V/-12V.
In the second case the components are supplied with +5V.

Normally the ISER8 is equipped with a ±12 V DC/DC converter.

If the types of piggy-backs on the ISER8 should be changed at any time, this may be done only with respect for the permitted combinations as described above. Otherwise malfunctions or damages on the ISER8 or on the piggy-backs cannot be excluded!
(See also section: Jumpers of the Serial Interfaces)

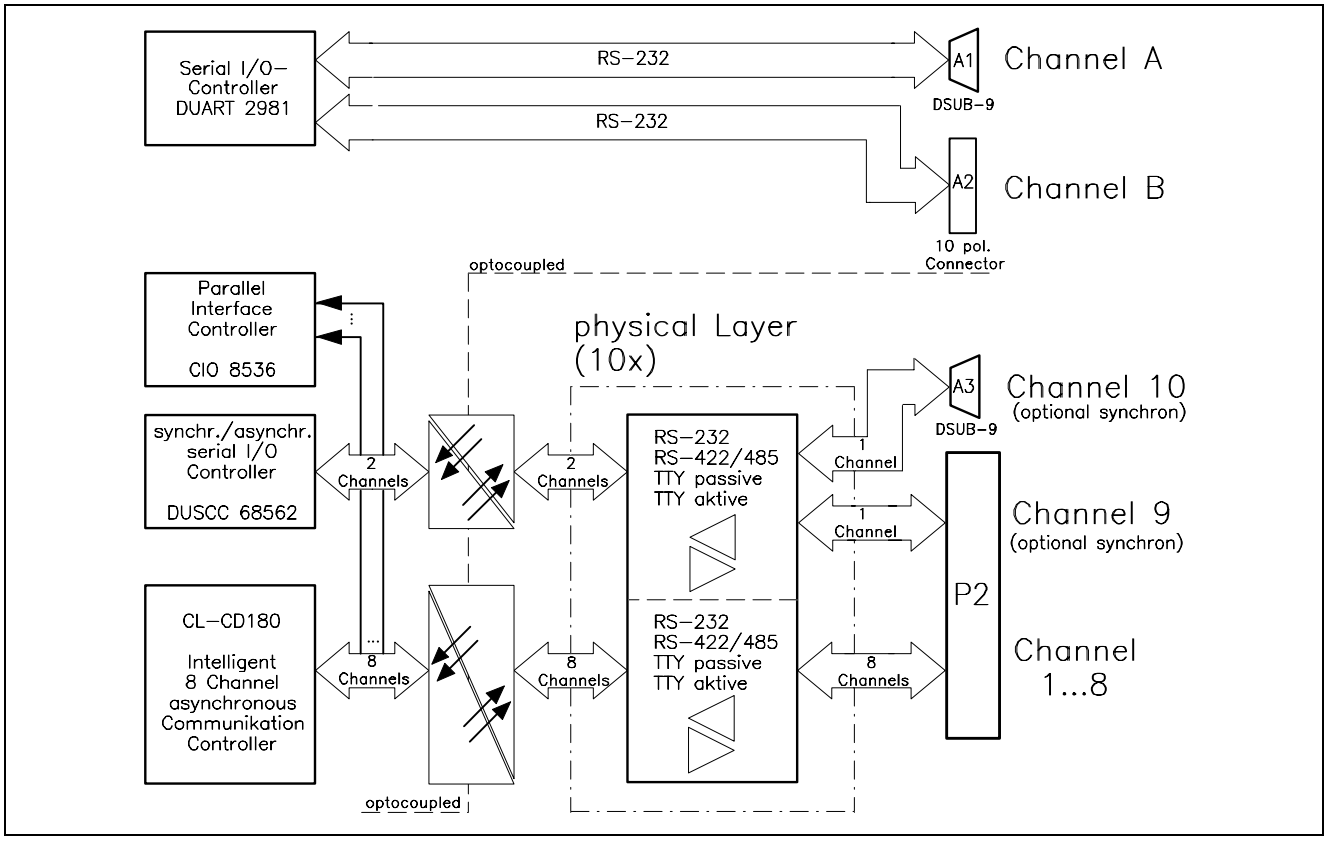


Fig. 3: Assignment of the Serial Channels to the Controllers and Connectors

1.9.2 Connection Diagrams of the Serial Interfaces

In the following the wiring of the serial interfaces is displayed with respect to the data direction. The figures shall explain the short designations of the signals used in the appendix (connector pin assignment). Moreover, from the appendix (circuit diagrams) the circuit diagrams of the different available piggy-backs are can be obtained.

1.9.2.1 The RS-232 Interface

When equipped with the driver components MAX232 (or equivalent), the channels 1 to 10 can be operated as RS-232 interfaces.

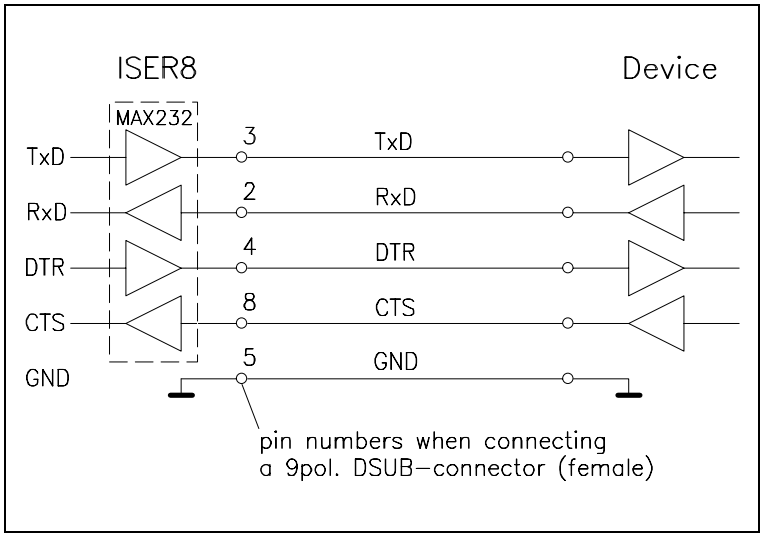


Fig. 4: Connection Diagram for RS-232 Operation

1.9.2.2 The RS-422 Interface

When equipped with the corresponding piggy-backs, the channels 1 to 10 can be operated as RS-422 interfaces.

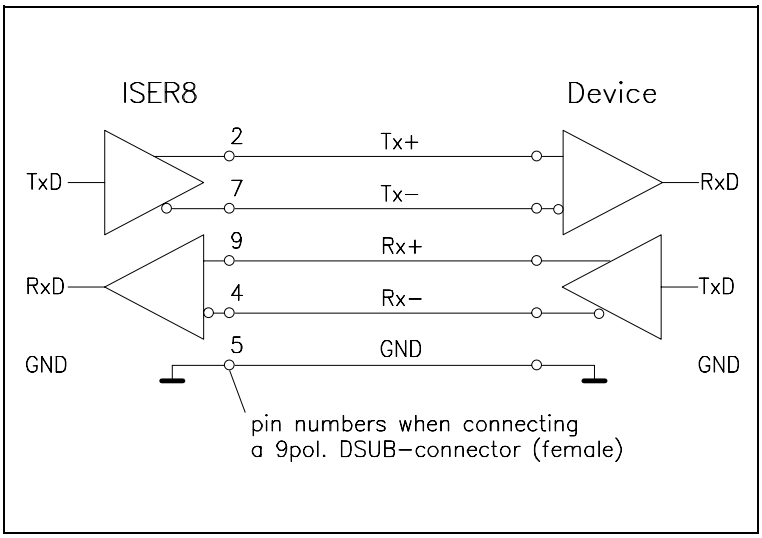


Fig. 5: Connection Diagram for RS-422 Operation

1.9.2.3 The RS-485 Interface

When equipped with the corresponding piggy-backs, the channels 1 to 10 can be operated as RS-485 interfaces.

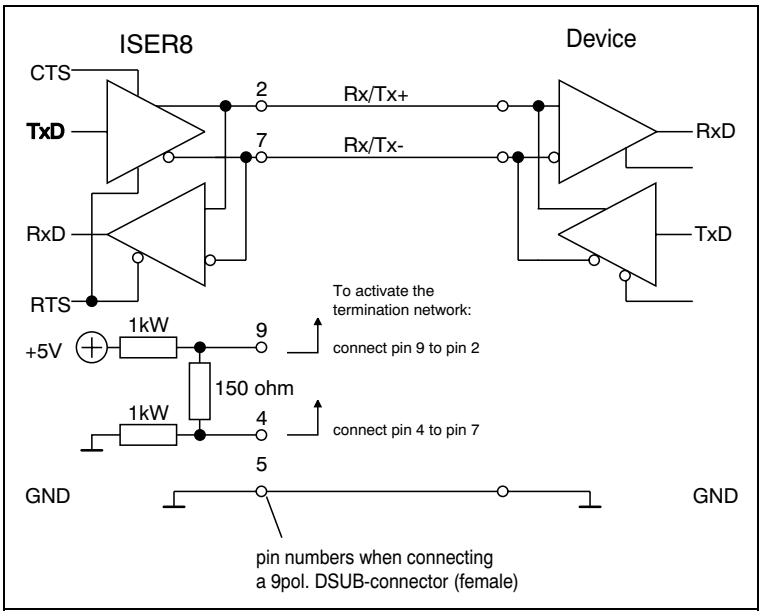


Fig. 6: Connection Diagram for RS-485 Operation

1.9.2.4 The TTY(20mA) Interface

When equipped with the corresponding piggy-backs, the channels 1 to 10 can be operated as 'passive' or as 'active' TTY interfaces.

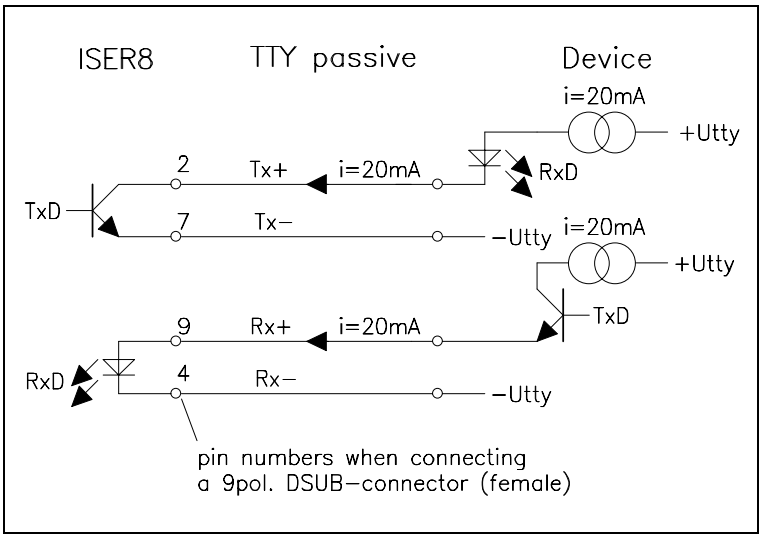


Fig. 7: Connection Diagram for TTY Operation (passive)

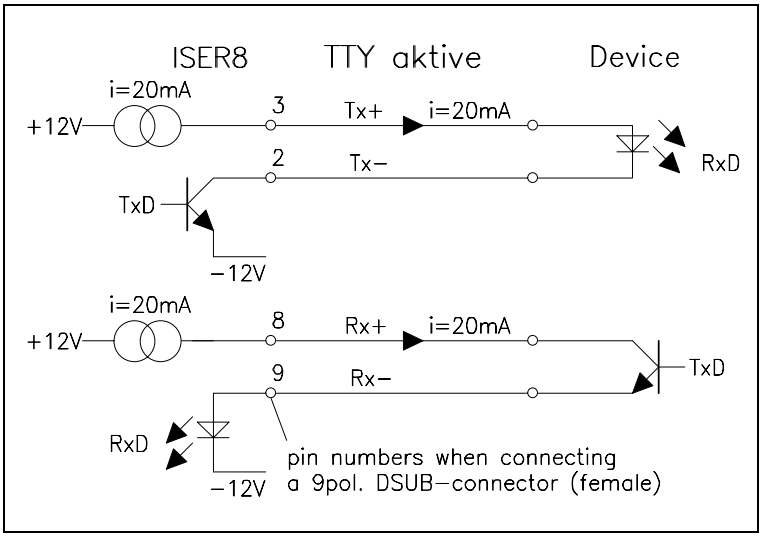
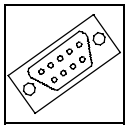


Fig. 8: Connection Diagram for TTY Operation (active)



2. Appendix

2.1 Connector Pin Assignments

2.1.1 VMEbus P1

pin	row a	row b	row c
1	DATA 0	BBSY*	DATA 8
2	DATA 1	BCLR*	DATA 9
3	DATA 2	ACFAIL*	DATA 10
4	DATA 3	BG0 IN*	DATA 11
5	DATA 4	BG0 OUT*	DATA 12
6	DATA 5	BG1 IN*	DATA 13
7	DATA 6	BG1 OUT*	DATA 14
8	DATA 7	BG2 IN*	DATA 15
9	GND	BG2 OUT*	GND
10	SYSCLOCK	BG3 IN*	SYSFAIL*
11	GND	BG3 OUT*	BERR*
12	DS1*	BR0*	SYSRESET*
13	DS0*	BR1*	LWORD*
14	WRITE*	BR2*	AM5
15	GND	BR3*	ADDR 23
16	DTACK*	AM0	ADDR 22
17	GND	AM1	ADDR 21
18	AS*	AM2	ADDR 20
19	GND	AM3	ADDR 19
20	IACK*	GND	ADDR 18
21	IACKIN*	SERCLOCK	ADDR 17
22	IACKOUT*	SERDAT	ADDR 16
23	AM4	GND	ADDR 15
24	ADDR 7	IRQ7*	ADDR 14
25	ADDR 6	IRQ6*	ADDR 13
26	ADDR 5	IRQ5*	ADDR 12
27	ADDR 4	IRQ4*	ADDR 11
28	ADDR 3	IRQ3*	ADDR 10
29	ADDR 2	IRQ2*	ADDR 9
30	ADDR 1	IRQ1*	ADDR 8
31	- 12V	-	+ 12V
32	+ 5V	+ 5V	+ 5V

P1 connector according to DIN 41 612-C 96 / a+b+c

Signals with * : active low
Current rating : max 1.0 A per pin

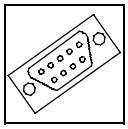
2.1.2 I/O Connector P2

P2 connector according to DIN 41612-C96 a+c

Pin	Reihe a						Reihe c					Pin
	RS-232	RS-485	RS-422	20mA passiv	20mA aktiv		RS-232	RS-485	RS-422	20mA passiv	20mA aktiv	
1	-	R/Tx-8	Tx-8	Tx-8	(-12V)		RxD8	R/Tx+8	Tx+8	Tx+8	Tx-8	1
2	CTS8	GND	GND	(I2+8)	Rx+8	KANAL 8	TxD8	-	-	(I1+8)	Tx+8	2
3	-	AΩ8+	Rx+8	Rx+8	Rx-8		DTR8	AΩ8-	Rx-8	Rx-8	(-12V)	3
4	RxD7	R/Tx+7	Tx+7	Tx+7	Tx-7		GND8	GND8	GND8	GND8	GND8	4
5	TxD7	-	-	(I1+7)	Tx+7		-	R/Tx-7	Tx-7	Tx-7	(-12V)	5
6	DTR7	AΩ7-	Rx-7	Rx-7	(-12V)	KANAL 7	CTS7	GND	GND	(I2+7)	Rx+7	6
7	GND7	GND7	GND7	GND7	GND7		-	AΩ7+	Rx+7	Rx+7	Rx-7	7
8	-	R/Tx-6	Tx-6	Tx-6	(-12V)		RxD6	R/Tx+6	Tx+6	Tx+6	Tx-6	8
9	CTS6	GND	GND	(I2+6)	Rx+6	KANAL 6	TxD6	-	-	(I1+6)	Tx+6	9
10	-	AΩ6+	Rx+6	Rx+6	Rx-6		DTR6	AΩ6-	Rx-6	Rx-6	(-12V)	10
11	RxD5	R/Tx+5	Tx+5	Tx+5	Tx-5		GND6	GND6	GND6	GND6	GND6	11
12	TxD5	-	-	(I1+5)	Tx+5		-	R/Tx-5	Tx-5	Tx-5	(-12V)	12
13	DTR5	AΩ5-	Rx-5	Rx-5	(-12V)	KANAL 5	CTS5	GND	GND	(I2+5)	Rx+5	13
14	GND5	GND5	GND5	GND5	GND5		-	AΩ5+	Rx+5	Rx+5	Rx-5	14
15	-	R/Tx-4	Tx-4	Tx-4	(-12V)		RxD4	R/Tx+4	Tx+4	Tx+4	Tx-4	15
16	CTS4	GND	GND	(I2+4)	Rx4+	KANAL 4	TxD4	-	-	(I1+4)	Tx+4	16
17	-	AΩ4+	Rx+4	Rx+4	Rx4-		DTR4	AΩ4-	Rx-4	Rx-4	(-12V)	17
18	RxD3	R/Tx+	Tx+3	Tx+3	Tx-3		GND4	GND4	GND4	GND4	GND4	18
19	TxD3	-	-	(I1+3)	Tx+3		-	R/Tx-3	Tx-3	Tx-3	(-12V)	19
20	DTR3	AΩ3-	Rx-3	Rx-3	(-12V)	KANAL 3	CTS3	GND	GND	(I2+3)	Rx+3	20
21	GND3	GND3	GND3	GND3	GND3		-	AΩ3+	Rx+3	Rx+3	Rx-3	21
22	-	R/Tx-2	Tx-2	Tx-2	(-12V)		RxD2	R/Tx+2	Tx+2	Tx+2	Tx-2	22
23	CTS2	GND	GND	(I2+2)	Rx2-	KANAL 2	TxD2	-	-	(I1+2)	Tx+2	23
24	-	AΩ2+	Rx+2	Rx+2	Rx2+		DTR2	AΩ2-	Rx-2	Rx-2	(-12V)	24
25	RxD1	R/Tx+1	Tx+1	Tx+1	Tx-1		GND2	GND2	GND2	GND2	GND2	25
26	TxD1	-	-	(I1+1)	Tx+1		-	R/Tx-1	Tx-1	Tx-1	(-12V)	26
27	DTR1	AΩ1-	Rx-1	Rx-1	(-12V)	KANAL 1	CTS1	GND	GND	(I2+1)	Rx+1	27
28	GND1	GND1	GND1	GND1	GND1		-	AΩ1+	Rx+1	Rx+1	Rx-1	28
29	RxD9	R/Tx+9	Tx+9	Tx+9	Tx-9		CLKI/O	CLK9	CLK9	-	-	29
30	TxD9	-	-	(I1+9)	Tx+9		-	R/Tx-9	Tx-9	Tx-9	(-12V)	30
31	DTR9	AΩ9-	Rx-9	Rx-9	(-12V)	KANAL 9	CTS9	GND	GND	(I2+9)	Rx+9	31
32	GND9	GND9	GND9	GND9	GND9		-	AΩ9+	Rx+9	Rx+9	Rx-9	32

()... The signals in parentheses are connected to the DSUB female, but are not necessary for the corresponding operation mode.

RS485: To activate the termination resistor array the signal AΩy+ has to be connected with R/Tx+y and the signal AΩy- has to be connected to R/Tx-y (y = 1, 2,...9).



2.1.3 I/O Connector P2 Junction Module Phoenix FLKM64 or FLKMS64

Pin	Reihe a						Reihe c					Pin
	RS-232	RS-485	RS-422	20mA passiv	20mA aktiv		RS-232	RS-485	RS-422	20mA passiv	20mA aktiv	
2	-	R/Tx-8	Tx-8	Tx-8	(-12V)		RxD8	R/Tx+8	Tx+8	Tx+8	Tx-8	1
4	CTS8	GND	GND	(I2+8)	Rx+8	KANAL 8	TxD8	-	-	(I1+8)	Tx+8	3
6	-	AΩ8+	Rx+8	Rx+8	Rx-8		DTR8	AΩ8-	Rx-8	Rx-8	(-12V)	5
8	RxD7	R/Tx+7	Tx+7	Tx+7	Tx-7		GND8	GND8	GND8	GND8	GND8	7
10	TxD7	-	-	(I1+7)	Tx+7		-	R/Tx-7	Tx-7	Tx-7	(-12V)	9
12	DTR7	AΩ7-	Rx-7	Rx-7	(-12V)	KANAL 7	CTS7	GND	GND	(I2+7)	Rx+7	11
14	GND7	GND7	GND7	GND7	GND7		-	AΩ7+	Rx+7	Rx+7	Rx-7	13
16	-	R/Tx-6	Tx-6	Tx-6	(-12V)		RxD6	R/Tx+6	Tx+6	Tx+6	Tx-6	15
18	CTS6	GND	GND	(I2+6)	Rx+6	KANAL 6	TxD6	-	-	(I1+6)	Tx+6	17
20	-	AΩ6+	Rx+6	Rx+6	Rx-6		DTR6	AΩ6-	Rx-6	Rx-6	(-12V)	19
22	RxD5	R/Tx+5	Tx+5	Tx+5	Tx-5		GND6	GND6	GND6	GND6	GND6	21
24	TxD5	-	-	(I1+5)	Tx+5		-	R/Tx-5	Tx-5	Tx-5	(-12V)	23
26	DTR5	AΩ5-	Rx-5	Rx-5	(-12V)	KANAL 5	CTS5	GND	GND	(I2+5)	Rx+5	25
28	GND5	GND5	GND5	GND5	GND5		-	AΩ5+	Rx+5	Rx+5	Rx-5	27
30	-	R/Tx-4	Tx-4	Tx-4	(-12V)		RxD4	R/Tx+4	Tx+4	Tx+4	Tx-4	29
32	CTS4	GND	GND	(I2+4)	Rx4+	KANAL 4	TxD4	-	-	(I1+4)	Tx+4	31
34	-	AΩ4+	Rx+4	Rx+4	Rx4-		DTR4	AΩ4-	Rx-4	Rx-4	(-12V)	33
36	RxD3	R/Tx+	Tx+3	Tx+3	Tx-3		GND4	GND4	GND4	GND4	GND4	35
38	TxD3	-	-	(I1+3)	Tx+3		-	R/Tx-3	Tx-3	Tx-3	(-12V)	37
40	DTR3	AΩ3-	Rx-3	Rx-3	(-12V)	KANAL 3	CTS3	GND	GND	(I2+3)	Rx+3	39
42	GND3	GND3	GND3	GND3	GND3		-	AΩ3+	Rx+3	Rx+3	Rx-3	41
44	-	R/Tx-2	Tx-2	Tx-2	(-12V)		RxD2	R/Tx+2	Tx+2	Tx+2	Tx-2	43
46	CTS2	GND	GND	(I2+2)	Rx2-	KANAL 2	TxD2	-	-	(I1+2)	Tx+2	45
48	-	AΩ2+	Rx+2	Rx+2	Rx2+		DTR2	AΩ2-	Rx-2	Rx-2	(-12V)	47
50	RxD1	R/Tx+1	Tx+1	Tx+1	Tx-1		GND2	GND2	GND2	GND2	GND2	49
52	TxD1	-	-	(I1+1)	Tx+1		-	R/Tx-1	Tx-1	Tx-1	(-12V)	51
54	DTR1	AΩ1-	Rx-1	Rx-1	(-12V)	KANAL 1	CTS1	GND	GND	(I2+1)	Rx+1	53
56	GND1	GND1	GND1	GND1	GND1		-	AΩ1+	Rx+1	Rx+1	Rx-1	55
58	RxD9	R/Tx+9	Tx+9	Tx+9	Tx-9		CLKI/O	CLK+9	CLK+9	-	-	57
60	TxD9	CLK-9	CLK-9	(I1+9)	Tx+9		-	R/Tx-9	Tx-9	Tx-9	(-12V)	59
62	DTR9	AΩ9-	Rx-9	Rx-9	(-12V)	KANAL 9	CTS9	GND	GND	(I2+9)	Rx+9	61
64	GND9	GND9	GND9	GND9	GND9		-	AΩ9+	Rx+9	Rx+9	Rx-9	63

()... The signals in parentheses are connected to the DSUB female, but are not necessary for the corresponding operation mode.

RS485: To activate the termination resistor array the signal AΩy+ has to be connected with R/Tx+y and the signal AΩy- has to be connected to R/Tx-y (y = 1, 2,...9).

2.1.4 Assignment of a 9 pole DSUB Female with the Signals of the Serial Channels 1...8

Signale					Pins		Signale				
					9-pol DSUB						
RS-232	RS-485	RS-422	20mA passiv	20mA aktiv			20mA aktiv	20mA passiv	RS-422	RS-485	RS-232
-	-	-	-	-	1	6	-	-	-	-	-
RxD	R/Tx+	Tx+	Tx+	Tx-	2	7	(-12V)	Tx-	Tx-	R/Tx-	-
TxD	-	-	(I1+)	Tx+	3	8	Rx+	(I2+)	GND	GND	CTS
DTR	AΩ-	Rx-	Rx-	(-12V)	4	9	Rx-	Rx+	Rx+	AΩ+	-
GND	GND	GND	GND	GND	5						

()... The signals in parentheses are connected to the DSUB female, but are not necessary for the corresponding operation mode.

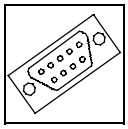
RS485: To activate the termination resistor array the signal AΩy+ has to be connected with R/Tx+y and the signal AΩy- has to be connected to R/Tx-y (y = 1, 2,...9).

2.1.5 Assignment of a 9 pole DSUB Female with the Signals of the Serial Channels 9..10 (optionally Synchronous Mode)

Signale					Pins		Signale				
					9-pol DSUB						
RS-232	RS-485	RS-422	20mA passiv	20mA aktiv			20mA aktiv	20mA passiv	RS-422	RS-485	RS-232
-	-	-	-	-	1	6	-	-	CLK+	CLK+	CLKI/O
RxD	R/Tx+	Tx+	Tx+	Tx-	2	7	(-12V)	Tx-	Tx-	R/Tx-	-
TxD	CLK-	CLK-	(I1+)	Tx+	3	8	Rx+	(I2+)	GND	GND	CTS
DTR	AΩ-	Rx-	Rx-	(-12V)	4	9	Rx-	Rx+	Rx+	AΩ+	-
GND	GND	GND	GND	GND	5						

()... The signals in parentheses are connected to the DSUB female, but are not necessary for the corresponding operation mode.

RS485: To activate the termination resistor array the signal AΩy+ has to be connected with R/Tx+y and the signal AΩy- has to be connected to R/Tx-y (y = 1, 2,...9).

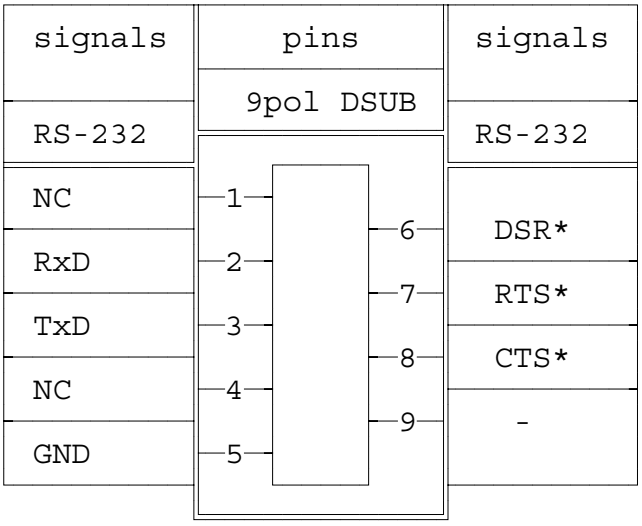


2.1.6 Serial Interfaces Port 1 and Port 2 (A1, A2) for Terminal Connection

The two serial interfaces comply with RS232C. The baud rate is 9600 Baud.

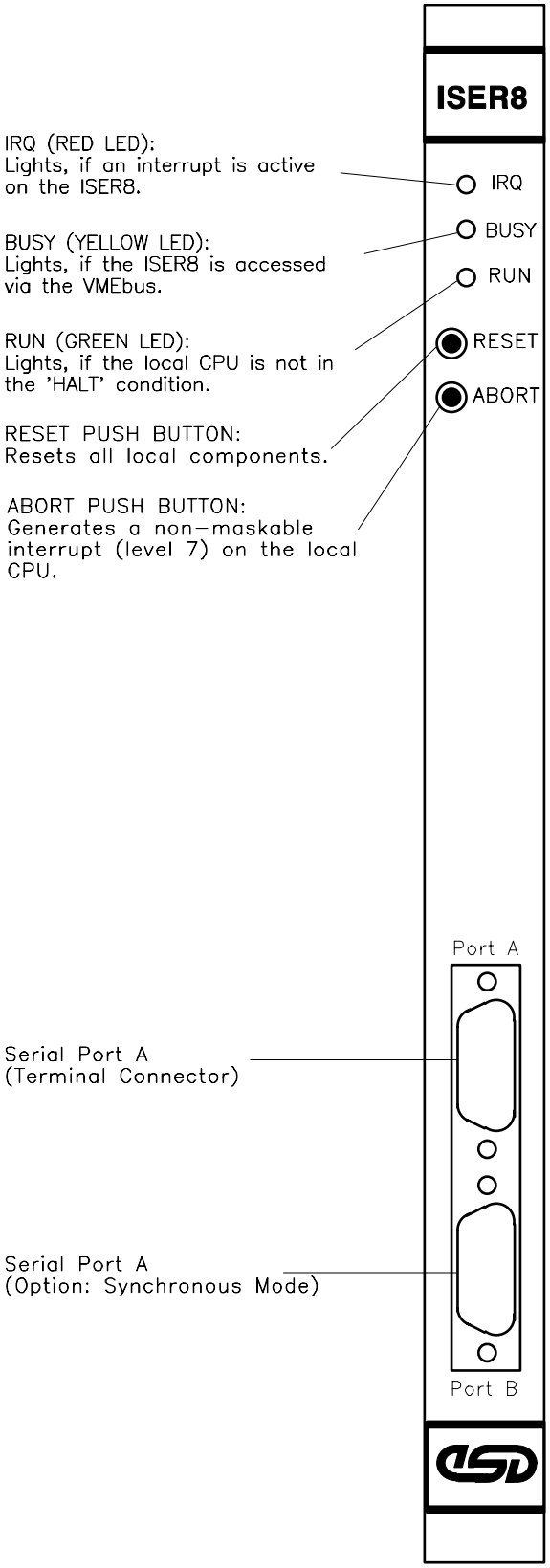
Port 1 is fed to a 9 pole DSub female (A1), and is designed for terminal connection. The DSUB female is located at the front panel (upper DSUB female).

Port 2 is covered only with Rxd, TxD and GND and is fed to a 10 pole connector plug (A2), which can be directly connected 1:1 as port 1 via a flatcable.



NC...not connected
*....These signals are only used via port 1 (A1).

2.2 Front Panel



2.3 Circuit Diagrams

2.4 Data Sheets

VME - ISER8

Intelligent Board for 10 serial Interfaces

Software Manual

N O T E

The information in this document has been carefully checked and is believed to be entirely reliable. **esd** makes no warranty of any kind with regard to the material in this document, and assumes no responsibility for any errors that may appear in this document. **esd** reserves the right to make changes without notice to this, or any of its products, to improve reliability, performance or design.

esd assumes no responsibility for the use of any circuitry other than circuitry which is part of a product of **esd** gmbh.

esd does not convey to the purchaser of the product described herein any license under the patent rights of **esd** gmbh nor the rights of others.

esd electronic system design gmbh
Vahrenwalder Str. 205
30165 Hannover
Germany

Phone: +49-511-372 98-0
Fax: +49-511-372 98-68
E-mail: info@esd-electronics.com
Internet: www.esd-electronics.com

USA / Canada
7667 W. Sample Road
Suite 127
Coral Springs, FL 33065
USA

Phone: +1-800-504-9856
Fax: +1-800-288-8235
E-mail: sales@esd-electronics.com

Document file:	I:\texte\Doku\MANUALS\VME\ISER8\Englisch\ISER8_01s.en6
Date of print:	14.11.2000

Described Firmware-Version:	isers50b

Changes in the Chapters

The changes in the user's manual listed below effect changes in the **hardware**, as well as changes in the **description** of the facts only.

Chapter	Changes versus previous version
-	Division in hardware and software maual

Further technical changes are subject to change without notice.

Content

1. Introduction

1.1 General

This manual describes the serial VMEbus interface boards **VME-ISER8** and **VME-ISER12**.
A large part of the descriptions is valid for the VME-ISER8 and VME-ISER12 board. In the following both boards are summarized under the concept **VME-ISER**.
Special data which concern only one of these boards are pointed to in corresponding places.

The VME-ISER8 is an intelligent interface board for the VMEbus, which locally supervises 8 asynchronous and 2 optionally synchronous or asynchronous serial interfaces.

The VME-ISER12 has got the same number of interfaces as the VME-ISER8. Two transition modules of type ESP360 can optionally be attached to VME-ISER12. In coherence with these modules the VME-ISER12 offers 10 asynchronous and 2 synchronous/asynchronous serial interfaces.

The user operates to a linear memory and is relieved of I/O supervision tasks by the local CPU.

The memory accessible to the user is organized in so-called channels, which consist of a header and a data range. The length of a channel amounts to 256 bytes (128 bytes net data), or 1024+128 bytes * (1 kbyte net data).

The structure of the header is identical for all occurring types of channels, the different channels differ in corresponding entries in the header of the channel.

The status of the serial interfaces and the setting of the serial interfaces parameters is transparently readable, resetting of the parameter ensues synchronously to the I/O transfer.

1.2 Channel Overview

1.2.1 Channel Types

The system consists of following types of channels:

- the parameter channels	1 channel per serial interface
- the data channel	1 receive channel (1 kbyte) 1 transmit channel (1 kbyte) 26 transmit channels (128 bytes each)
- the interrupter channel	1 channel per board

Channels are software structures, which are chained by pointers.
The 'ROOT pointer', as well as a 'Card Id' are at fixed addresses.

1.2.2 Tasks of the VME Master Servers

The VME master server for the serial interfaces must essentially fulfill the following tasks:

- Search a free channel and occupy this channel

- Entry of the transfer mode

- (Data transfer to the VME-ISER memory for transmit operation)

- Activation of the slave server (local interrupt generation)

- Polling on 'ready' or reactivation by VME interrupt

- (Data transfer from the VME-ISER memory for receive operation)

- Channel enable

1.3 Initialization of the System

In the following all addresses are indicated relatively to the card base address and must be addressed correspondingly by the VME master CPU.

After a system reset the local CPU initializes its local memory and rebuilds the channel pointer chain. This can take up to 2 sec depending on the memory size. After a restart the master CPU should check the following entries:

- read access to the base address of the slave board.
  If the board responds with a 'DTACK signal', it is physically available at the correspondent address; otherwise a 'BUSERROR' occurs (e.g. via time- out) because the board is not available
  >> abort of the initialization.

- check of the address **CPUID** = $0998 to: $49534552.L
  The local CPU must have an ASCII entry: "ISER" =($49, $53, $45, $52).

- check of address **ANCHOR** = $099C to unequal to $0
  The local CPU inserts the ROOT pointer at the buffer structure (default: $00008000.L)

The local CPU has now built up the buffer structure described in the following, which enables a communication with the master CPU.

1.4 The Channel Structure

1.4.1 Chaining of the Channels

All channels are chained by pointers, where it must be distinguished between a memory chaining and a forward/backward chaining.
The memory chaining connects all available channels, while the forward/backward chaining only connects those channels related to the corresponding interface.

Memory Chaining:

Sequential chaining
The root pointer to the first available channel is a longword at the address **ANCHOR** =$0099C, the pointer to the next channel (forward pointer) is a longword each time in the location ***iofor*** of the channel header. The forward pointer of the last channel points back to the first channel.
As default **ANCHOR** is set to $00008000. All addresses listed in the tables refer to this base, but are relocatable without restrictions.
The length of a channel normally is 256 bytes and is divided into 128 bytes of header and 128 bytes of data.

Star-shaped chaining (from Software-Rev. iser 50b)
The star-shaped chaining speeds up the snapping of the addresses of the channels. In the interrupt channel the successive addresses of all parameter channels can be found. In every parameter channel the addresses of the assigned Tx- and Rx-channels are stored.

Note:
The sequential chaining and the star-shaped chaining are both available and can be used alternatively.

Address Offset HEX	+0	+2	+4	+6	+8	+A	+C	+E
00	iofor		ioback		iotyp		ioname	
10	sema / iostat	iocmmd	ionext		iolev / iovec		iobnum	iolen
20	iobuff		iorecl	iostio / ioldn	iomode	iotout / iodrv	iofnam...	
30	...iofnam...							
40	...iofnam		ioentr		iotent			iofree
50	iorxin		iofree...					
60	...iofree...							
70								...iofree
80	iodata...							
90	...iodata...							
A0	...iodata...							
B0	...iodata...							
C0	...iodata...							
D0	...iodata...							
E0	...iodata...							
F0								...iodata

User read/write cells

User read-only cells

User don't care

Table 1.4.1: Internal Channel Structure with READ/WRITE Assignment of the Cells

1.4.2 Description of the individual Channel Locations

Summary of the channel locations

Name	Offset [HEX]	Organization	Description	Default/Preset
iofor	00	longword	Pointer to next channel	
ioback	04	longword	not used	$0000000
iotyp	08	word	channel type (see below)	
ioname	0A	6 byte ASCII	channel identifier as character string	
iosema	10	byte	channel semaphore	preset: $00
iostat	11	byte	channel status	preset: $00
iocmmd	12	word	channel command	preset: $0000
ionext	14	longword	forward / backward pointer to next channel	
ioilev	18	byte	VME-Irq-Level for Slave-Irq	
ioivec	19	byte	VME-Irq-Vektor for Slave-Irq	
iobnum	1A	word	number of the specific channel type	
iolen	1C	longword	length of the data range	
iobuff	20	longword	pointer to data range	
iorecl	24	word	number of the data in the data range	
iostio	26	byte	I/O status	default: $00
ioldn	27	byte	Interface no. 1 ... 10	
iomode	28	word	transmit / receive mode	
iotout	2A	byte	time-out	
iodrv	2B	byte	reserved	
iofnam	2C ... 43	ASCII	reserved	default: $0000
ioentr	44	longword	pointer to user protocol (only parameter channel)	
iotent	48	longword	reserved for Tx-server	
iofree	4C ... 7F		reserved	default: $0000
iorxln	50	word	number of received data	
iodata	80 ... FF	byte	data range (128 byte channels)	
	80 ... 47F	byte	data range (1 Kbyte-channels)	

Table 1.4.2: Description of the Channel Cells

Explanation of the individual channel cells

iofor supports the memory chaining of the channels. *iofor* always points to the start address of the next channel, *iofor* of the last channel points to the first channel again.

ioback points to the start address of the preceding channel

iotyp is the channel identifier and distinguishes the following channel types:
- $FFFF interrupter channel
- $000C parameter-channel
- $0014 default channel (not used)
- $0018 buffer
- $001C buffer-channel (not used)
- $0114 Tx-buffer long
- $0214 Rx-buffer long

ioname contains the channel identifier as a 6 bytes ASCII string and a consecutive numbering:

- *Irch*	interrupter channel	
- *PARAxy*	parameter channel	with xy = 01, 02, ... 09, 0A
- *TBUFxy*	transmit buffer_long	with xy = 01, ... 0A
- *RBUFxy*	receive buffer_long	with xy = 01,... 0A
- *Buffxz*	transmit buffer (128 byte)	with x = 1, ... A z = a, b,...z

iosema is covered with the channel semaphore and with the channel status bit:

- Bit 7	semaphore:	'0' -- channel is free
		'1' -- channel is occupied
- Bit 6 - 1	reserved, default: '0'	
- Bit 0	channel status:	'0' -- channel is busy
		'1' – channel is ready

iostat is not yet supplied and is preset to $0

iocmmd is the channel command and is only necessary for setting the interface parameters (see interface parameter setting, from page 19).

ionext is the pointer to the next data channel. Is only used for data channels, otherwise 0.

ioilev und ***ioivec***
determine the slave interrupt behaviour. If *ioilev* and *ioivec* = 0, then the slave will not generate an interrupt at the end of the instruction corresponding to the channel, but only *iosema* is set analogously. Otherwise an IRQ on the VMEbus with the IRQ level *ioilev* (1..7) will be generated by the IRQ vector *ioivec* ($00 .. $FF).

iobnum contains the consecutive numbering of the channels.
For the interrupter channel *iobnum* has a value of 0.

iolen contains the available data buffer length. If the data buffer is located within the channel structure (default), then *iolen* = $00080 == 128 bytes, or $400 respectively. External data may have an unlimited length.

iobuff is the pointer to the data buffer of the corresponding pointer channel. As default *iobuff* points to *iodata*. At external data buffers *iobuff* may point to any local address, so that addressing the data buffer **must** use the actual content of *iobuff*!

iorecl determines the number of valid data in the data range. (number of data to be sent or received).

If *iorecl* is negative, i.e. the MSB is set, the transmission has been stopped with error!

error codes:
- $8007 - time-out
- $801E - framing error
- $801F - overrun error
- $8020 - parity error
- $8046 - break detected

iostio is not yet supplied and is preset to $00.

ioldn contains the channel server no. (1,...,10)

iomode supports the setting of the data direction (transmit/receive operation) as well as setting the receiving protocol parameters:

Bit-No	Mnemo	Description	
15	*MODBWA*	0	After transmission of all data **no** IRQ will be generated, the requested channel will automatically be released again by the slave
		1	After transmission of all data *ready* will be set *iosema*, or the indicated IRQ will be generated respectively. The requested channel will **not** be released by the slave.
14	*MODBOU*	0	Identification: receive channel
		1	Identification: transmit channel
13	*MODBOU*	1	After detection of a <cr> ($0D) the reception of this channel will be terminated.
12	*MODBLF*	1	After detection of a <lf> ($0A) the reception of this channel will be terminated.
11	*MODBEO*	1	After detection of a <eot> ($04) the reception of this channel will be terminated.
10	*MODBSC*	-	suppress_command: actually not connected
9	*MODBNE*	-	no_echo: actually not connected
8	*MODBIN*	0	no binary transfer
		1	binary transfer: no end check, no software-handshake

Table 1.4.3: Bits of *iomode*

Bit 7-0 of *iomode* are reserved as mode extension bits. The following combinations are already defined:

- $00 normal I/O transfer (default)
- $08 only for receive operation:
  All characters in the local buffers will be deleted.

iotout time-out value
The MSB (bit 7) enables the *Time_Out* supervision of the channel.
If no transfer into an active channel buffer occurs, after the time *T_Out* the channel will be released and the status *Time_Out* is returned! (via *iorecl*).

iofnam is reserved for ASCII entries (up to 24 bytes).
Actually following entry will be evaluated:
On the ASCII string SCAN in the first 4 bytes of *iofnam* the following return conditions are valid for a receive channel:

1.) Return of the buffer, if <*iorecl*> data have been received
2.) Return of the buffer, if one of the end conditions specified in <*iomode*> is valid.
3.) Return of the buffer, if no more data are available in the local interrupt buffer, i.e. if the interrupt buffer is empty, the receive channel is returned immediately with <*iorecl*>=0.

For all other entries into *iofnam* only the end conditions 1.) and 2.) are valid. With the entry PROT data are received via a special user protocol.

ioentr supports the embedding of an user-specific receive protocol (only parameter channel). The start address of a protocol loaded into a free memory area is registered here.

iotent is reserved for embedding of a user-specific transmit protocol (only parameter channel).

iorxln determines the number of valid received data, specially in the error case.

iofree is actually not used and is preset to $00.

iodata is the default data buffer of a channel and has a length of 128 bytes, or 1 kbyte respectively (*TBUFxy, RBUFxy*).
Writing to memory out of the data buffer limits will destroy the I/O structure!

1.5 Data Channel Management

1.5.1 General

As mentioned above, the channels are divided into parameter channels, buffer channels, default channels and interrupter channels. To each serial interface a parameter (TX) buffer, a default Tx buffer, an Rx buffer, and a number of buffers of the 'Buffer-Pool' are allocated.

The parameter buffer, the Tx buffer and the Rx buffer are **exclusively** allocated to the corresponding interface. As a principle the buffers may be used by any channel. The pointer chaining results in a priorized buffer allocation to the corresponding interface channels.

The chaining of the TX buffers and of the buffer channels is displayed in the following tables. The forward/backward pointer *ionext* allocates the corresponding Tx buffer channel to a buffer. The *ionext* pointer of the last buffer points to the Tx buffer again.

This channel distribution has been chosen for a very flexible memory allocation, while the searching algorithm remains quick and simple.

1.5.2 Overview to the Channels with Chaining via Pointer

Channel Root Pointer		
Address [HEX]	Content [HEX]	Remarks
0099C	08000	Start address of the buffer range

Table 1.5.1: Channel Root Pointer to Address *ANCHOR*

Buffer Number [DEZ]	Address [HEX]	Channel Header						Remarks
		iofor [HEX]	iobnum [HEX]	ioldn	ionext [HEX]	iolen [HEX]	ionam	
0	08000	08100	0	0	0	80	Irch__	interrupter channel
1	08100	08200	1	1	0	80	PARA01	parameter channel 1
2	08200	08300	2	2	0	80	PARA02	parameter channel 2
3	08300	08400	3	3	0	80	PARA03	parameter channel 3
4	08400	08500	4	4	0	80	PARA04	parameter channel 4
5	08500	08600	5	5	0	80	PARA05	parameter channel 5
6	08600	08700	6	6	0	80	PARA06	parameter channel 6
7	08700	08800	7	7	0	80	PARA07	parameter channel 7
8	08800	08900	8	8	0	80	PARA08	parameter channel 8
9	08900	08A00	9	9	0	80	PARA09	parameter channel 9
10	08A00	08B00	A	10	0	80	PARA0A	parameter channel 10

Table 1.5.2: Interrupter Channel and Parameter Channels

Buffer Number [DEZ]	Address [HEX]	Channel Header						Remarks
		iofor [HEX]	iobnum [HEX]	ioldn	ionext [HEX]	iolen [HEX]	ionam	
11	08B00	08F80	B	1	0E500	400	TBUF01	transmit buffer 01
12	08F80	09400	C	1	08F80	400	RBUF01	receive buffer 01
13	09400	09880	D	2	0FF00	400	TBUF02	transmit buffer 02
14	09880	09D00	E	2	09880	400	RBUF02	receive buffer 02
15	09D00	0A180	F	3	11900	400	TBUF03	transmit buffer 03
16	0A180	0A600	10	3	0A180	400	RBUF03	receive buffer 03
17	0A600	0AA80	11	4	13300	400	TBUF04	transmit buffer 04
18	0AA80	0AF00	12	4	0AA80	400	RBUF04	receive buffer 04
19	0AF00	0B380	13	5	14B00	400	TBUF05	transmit buffer 05
20	0B380	0B800	14	5	0B380	400	RBUF05	receive buffer 05
21	0B800	0BC80	15	6	16700	400	TBUF06	transmit buffer 06
22	0BC80	0C100	16	6	0BC80	400	RBUF06	receive buffer 06
23	0C100	0C580	17	7	18100	400	TBUF07	transmit buffer 07
24	0C580	0CA00	18	7	0C580	400	RBUF07	receive buffer 07
25	0CA00	0CE80	19	8	19B00	400	TBUF08	transmit buffer 08
26	0CE80	0D300	1A	8	0CE80	400	RBUF08	receive buffer 08
27	0D300	0D780	1B	9	1B500	400	TBUF09	transmit buffer 09
28	0D780	0DC00	1C	9	0D780	400	RBUF09	receive buffer 09
29	0DC00	0E080	1D	10	1CF00	400	TBUF0A	transmit buffer 0A
30	0E080	0E500	1E	10	0E080	400	RBUF0A	receive buffer 0A

Table 1.5.3: Transmit and Receive Buffer

Buffer Number [DEZ]	Address [HEX]	Channel Header						Remarks
		iofor [HEX]	iobnum [HEX]	ioldn	ionext [HEX]	iolen [HEX]	ionam	
31	0E500	0E600	1F	1	0E600	80	BUFF1a	26 buffer for channel 1
32	0E600	0E700	20	1	0E70	80	BUFF1b	
:	:	:	:	:	:	:	:	
55	0FD00	0FE00	37	1	0FE00	80	BUFF1y	
56	0FE00	0FF00	38	1	08800	80	BUFF1z	
57	0FF00	10000	39	2	10000	80	BUFF2a	26 buffer for channel 2
:	:	:	:	:	:	:	:	
82	11800	11900	52	2	09400	80	BUFF2z	
83	11900	11A00	53	3	11A00	80	BUFF3a	26 buffer for channel 3
:	:	:	:	:	:	:	:	
108	13200	13300	6C	3	09D00	80	BUFF3z	
109	13300	13400	6D	4	13400	80	BUFF4a	26 buffer for channel 4
:	:	:	:	:	:	:	:	
134	14C00	14D00	86	4	0A600	80	BUFF4z	
135	14D00	14E00	87	5	14E00	80	BUFF5a	26 buffer for channel 5
:	:	:	:	:	:	:	:	
160	16600	16700	A0	5	0AF00	80	BUFF5z	
161	16700	16800	A1	6	16800	80	BUFF6a	26 buffer for channel 6
:	:	:	:	:	:	:	:	
186	18000	18100	BA	6	0B800	80	BUFF6z	
187	18100	18200	BB	7	18200	80	BUFF7a	26 buffer for channel 7
:	:	:	:	:	:	:	:	
212	19A00	19B00	D4	7	0C100	80	BUFF7z	
213	19B00	19C00	D5	8	19C00	80	BUFF8a	26 buffer for channel 8
:	:	:	:	:	:	:	:	
238	1B400	1B500	EE	8	0CA00	80	BUFF8z	
239	1B500	1B600	EF	9	1B600	80	BUFF9a	26 buffer for channel 9
:	:	:	:	:	:	:	:	
264	1CE00	1CF00	108	9	03D00	80	BUFF9z	
265	1CF00	1D000	109	10	1D000	80	BUFFAa	26 buffer for channel 10
:	:	:	:	:	:	:	:	
289	1E700	1E800	121	10	1E800	80	BUFFAy	
290	1E800	08000	122	10	0DC00	80	BUFFAz	

Table 1.5.4: Buffer Channels 1 to 10

1.6 Buffer Allocation

1.6.1 Memory Allocation via Semaphore

For a multitasking and multiuser memory management the memory allocation ensues via a semaphore, which can be accessed by the indivisible assembler command TAS.

Beginning with the corresponding default channel the semaphore of the channels is occupied.

On a successful access the corresponding channel is occupied. If not, the next buffer must be determined by *ionext*. Abort and wait conditions may be a certain number of unsuccessful accesses or the detection of 'wrap-around' (new_pointer < old_pointer).

After executing the I/O instruction either the slave server returns the channel by releasing the semaphore or the master must decide, when the channel will be available again.

1.6.2 Example of a Buffer Allocation

```
* Allocate memory on ISER-8/ISER-12
*
            MOVEA.L crdadr,A0       ;Base address ISER-8/ISER12
            MOVE.L  dfltbf,D0       ;buffer address relative
                                    ;to default address
            BSR     srchbuff        ;forward/backward buffer
            BNE     no_success      ;no buffer available
*           sonst:  in A0 actual absolute address of the channel
*                   in D0 buffer address relative to base address
            ....
            ....
         -- Transfer --
            END

srchbuff    MOVE.L  D0,D1                   ;end address(e.g. to start
                                            ;address as final condition)
srch1       TAS     iosema(A0,D0.L)         ;Semaphore access
            BEQ.S   srchex                  ;Semaph. was not occupied
                                            ;buffer address in D0
            MOVE.L  ionext(A0,D0.L),D0      ;next channel
            CMP.L   D0,D1                   ;end condition ?
            BGT.S   srch1                   ;No, go ahead searching
            TST.L   D0                      ;flag 'NE'
srchex      LEA     0(A0,D0.L),A0           ;absolute address in A0
            RTS
```

2. Channel Description

2.1 Description of the Data Channels

Data channels serve for the transfer of transmitted/received data and are of the type default channel or buffer channel. Before the beginning of a transmit/receive transfer a data channel has to be allocated according to the example above. Then the header of the channel is supplied with the corresponding parameters, if necessary data are input and are handed over to the local CPU.

	Address Offset HEX	+0	+2	+4	+6	+8	+A	+C	+E
HEADER	00	*iofor*		*ioback*		*iotyp*		*ioname*	
	10	*sema* / *iostat*	*iocmmd*	*ionext*		*iolev* / *iovec*		*iobnum*	*iolen*
	20	*iobuff*		*iorecl*	*iostio* / *ioldn*	*iomode*	*iotout* / *iodrv*		*iofnam...*
	30	*...iofnam...*							
	40	*...iofnam*		*ioentr*		*iotent*			*iofree*
	50	*iorxlen**		*iofree...*					
	60	*...iofree...*							
	70								*...iofree*
DATA AREA	80	*iodata...*							
	90	*...iodata...*							
	A0	*...iodata...*							
	B0	*...iodata...*							
	C0	*...iodata...*							
	D0	*...iodata...*							
	E0	*...iodata...*							
	F0	*...iodata*							

* only for Rx-Buffer

Table 2.1.1: Internal Channel Structure (valid for all types of channels)

Address Offset HEX	+0		+2		+4		+6		+8		+A		+C		+E	
8B00	00	00	**8F**	**80**	00	00	**8A**	**00**	**01**	**14**	**'TBUF01'**					
8B10	00	00	00	00	00	00	**E5**	**00**	00	00	00	**0B**	00	00	**04**	**00**
8B20	00	00	**8B**	**80**	00	00	00	**01**	00	00	00	00	00	00	00	00
8B30	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
8B40	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
8B50	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
8B60	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
8B70	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

8B80	*iodata...*
:	*...iodata...*

8F70	*...iodata*

Table 2.1.2: Default Channels (example: TBUF01)

2.2 Description of the Parameter Channel

2.2.1 Structure of the Parameter Channel

To each serial interface channel a so-called parameter channel is assigned. In the data range of this parameter channel the actual status of the interface is stored, which can be read completely transparently by the VME master.

The parameter channel is also necessary for the parameterization of the interface. For this the actual parameters are input at the corresponding sections of the parameter structure and the parameter channel is handed over to the VME-ISER server as 'transmit channel' (see also: ‘output channels’, on page 31). By this a synchronization with running transmit and receive jobs can be achieved.

The parameter structure is separated into 2 different parts:
- parameters, which can be written to by the user (offset: $80 - $BF)
- parameters, which can **only be read** by the user (offset: $C0 - $FF)

The parameters *txb...hnd* are formatted as byte and can be interpreted as identifiers for the physical parameterization.

Address Offset HEX	+0		+2		+4		+6		+8		+A		+C		+E	
8100	00	00	**82**	**00**	00	00	00	00	00	**0C**	***'PARA01'***					
8110	00	00	*'iocmmd'*		00	00	00	00	00	00	00	**01**	00	00	00	**80**
8120	00	00	**81**	**80**	00	00	00	00	00	00	00	00	00	00	00	00
8130	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
8140	00	00	00	00	*Protokoll*				*Protokoll*				00	00	00	00
8150	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
8160	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
8170	*tx_buffer1*				*rx_buffer1*				00	00	00	00	00	00	00	00
8180	*txbs*	*rxbs*	*chrls*	*stpls*	*parts*	*hnds*	*rxtime0*		*rxtime1*		*ttimes*		*txclkmods*	*rxclkmods*	reserved	
8190	*txbvs*				*rxbvs*				*protoks*	*encodes*	00	00	00	00	00	00
81A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
81B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
81C0	*txb*	*rxb*	*chrl*	*stpl*	*part*	*hnd*	*rtime0*		*rtime1*		*ttime*		*txclkmod*	*rxclkmod*	reserved	
81D0	*txbv*				*rxbv*				*protok*	*encode*	*endpar= FFFF*		00	00	00	00
81E0	*rxfifo*	*rxtout*	*resrv*	*spchr1*	*spchr2*	*spchr3*	*spchr4*	00	00	00	00	00	00	00	00	00
81F0	*txstat*	*rxstat*	*errlog*	00	00	00	00	00	00	00	00	00	00	00	00	00

Table 2.2.1: Parameter Channels (example: parameter Channel 1)

2.2.2 Description of the Parameter

Write accesses to the parameters can only ensue, if in the element *iocmmd* the command *paraxy* ($0000) is entered. Read accesses to the parameters are always possible, independently from *iocmmd*. (see also 'Command Transfer via the Parameter Channel', on page 25).

Writeable and readable parameters:

txbs	Index desired value *baud*:	transmitter baud rate
rxbs	Index desired value *baud*:	receiver baud rate
chrls	Index desired value *chri*:	bits/char
stpls	Index desired value *stpi*:	number of stop bits
parts	Index desired value *pari*:	parity type
hnds	Index desired value *hndi*:	handshake mode

Assignment of the Parameter indices:

```
Meaning of the index baud:
0  --   baud rate = 38400
1  --   baud rate = 19200
2  --   baud rate =  9600
3  --   baud rate =  4800
4  --   baud rate =  2400
5  --   baud rate =  1200
6  --   baud rate =   600
7  --   baud rate =   300
8  --   baud rate =   150
9  --   baud rate =   110
10 --   baud rate =    75
11 --   baud rate =    50
$FF --  baud rate = variable via txbv, rxbv

only for channel 9 and 10:
12  --   baud rate =  76800
13  --   baud rate = 115200

Meaning of the index chri:
0  --   8 bits per character
1  --   7 bits per character
2  --   6 bits per character
3  --   5 bits per character

Meaning of the index stpi:
0  --   1 stop bit
1  --   2 stop bits

Meaning of the index pari:
0  --   no Rx parity, no Tx parity
1  --   Rx/Tx parity ODD
2  --   Rx/Tx parity EVEN

Meaning of the index hndi:
0  --   hardware handshake DTR/CTS
1  --   software handshake XON/XOFF
2  --   modem operation RTS, CTS handshake
3  --   no handshake
4  --   RS-485 operation, no handshake
5  --   RS-422 operation, XON/XOFF handshake
```

rtime0s* Receive time-out for the first character in msec

0: Receive time-out disabled

rtime1s* Receive 'character to character' time-out in msec
0: no 'character to character' time-out

ttimes* Transmit Time-Out in msec
0: Transmit Time-Out disabled
* see also section 'Time-out' on page 32

rxclkmods Clock-mode of the DUSCC/SCC-channels has to be indicated separately for receive and transmit:

txclkmods

rxclkmods ***txclkmods***	*txbvs*	Mode	Function of the Pin RxTxCLK	Clock
x	0	Channel off	-	-
0	≠ 0	Async-Mode	-	16x baud rate
1	≠ 0	Synch-Mode	Pin RxTxCLK = OUT	1x baud rate
2	≠ 0	Synch-Mode	Pin RxTxCLK = OUT	16x baud rate
-1	≠ 0	Synch-Mode	Pin RxTxCLK = IN	1x baud rate
-2	≠ 0	Synch-Mode	Pin RxTxCLK = IN	16x baud rate

Table 2.2.2: Evaluation of *rxclkmods* and *txclkmods*

Pin RxTxCLK = DUSCC/SCC-Pin 39 (J3A-Pin 3) for channel 9,
or DUSCC/SCC-Pin 10 (J3-Pin 3) for channel 10

txbvs baud rate absolute, range of values 50 . . . ∞ (asynchronous),
rxbvs dimension baud

In *txbvs* and *rxbvs* the actual baud rate is indicated as absolute number. If a baud rate is desired, that deviates from the baud rates, which can be selected via *txb*, or *rxb*, via *txbvs*, or *rxbvs* the baud rates can be handed over as an absolute value (*txbs*, or *rxbs* set to $FF).
The interface is programmed with the nearest possible baud rate and the real value of the adjusted baud rate is handed back in *txbv* and *rxbv*.

Example: Parameter setting with Tx baud rate 115.000 baud at the VME-ISER8

Input : $FF --> *txbs*
Input : 115000 --> *txbvs*
Output: -->> *txbv* = 115200
(actual baud rate = 115200 baud!)

Note: The VME-ISER12 offers a better resolution for the setting of the absolute baud rate than the VME-ISER8, because of an additional fundamental frequency to generate the baud rate.

protoks Protocol mode of channel 9 and 10

protoks	Protocol mode
0	UART mode (all parameters of the parameter channels 9 and 10 are relevant)
1	HDLC mode (only the parameter of the channels 9 and 10, which are necessary for the synchronous transmission have to be considered: *rtime02, txclkmods, rxclkmods. txbvs, encode*)

Table 2.2.3: Protocol mode

encodes Signal coding of the serial Interfaces
Only the format NRZ (No Return to Zero) is supported (*encodes* = 0) at the moment.

Only readable parameter:

Following parameters serve as status information:
(**cannot** be written by the user !!)

txb	Index actual value *baud*:	transmitter baud rate
rxb	Index actual value *baud*:	receiver baud rate
chrl	Index actual value *chri*:	bits/character
stpl	Index actual value *stpi*:	number of stop bits
part	Index actual value *pari*:	parity type
hnd	Index actual value *hndi*:	handshake mode
	(assignment of the indices see page 20.)	

rtime0* Receive time-out for the first character in msec
rtime1* Receive 'character to character' time-out in msec
***ttime** ** Transmit time-out in msec
* see also section 'Time-out' on page 32

txclkmod, read parameter of the clock mode of the DUSCC/SCC channels
rxclkmod (Meaning of the parameter see Table on page 21)

txbv, baud rate absolute, range of values 50...38400, unit Baud
rxbv in *txbv* and *rxbv* the actual baud rate is indicated as an absolute number.
(see also above: 'txbvs', 'rxbvs' on page 21)

protok protocol mode of the channels 9 and 10
$00 - UART mode
$01 - HDLC mode
(see also ‘protoks’ on page 22)

encode signal coding of the serial Interfaces
Only the format NRZ (No Return to Zero) is supported (*encodes* = 0) at the moment .

rxfifo internal FIFO threshold for Rx interrupt (local !!)
rxtout time for Rx time-out in 5 msec units (local !!)
resrv reserved
spchr1-
spchr4 internal controller commands

txstat status of the transmitters

Bit 7 : not used
Bit 6 : not used
Bit 5 : not used
Bit 4 : not used
Bit 3 : '1' - Tx time-out occurred
'0' - no Tx time-out occurred
Bit 2 : '1' - Tx queue filled up
'0' - Tx queue ready
Bit 1 : '1' - transmitter disabled by handshake
'0' - transmitter enabled by handshake
Bit 0 : '1' - transmitter disabled
'0' - transmitter enabled

rxstat status of the receivers

Bit 7 : '1' - break recognized
'0' - no break recognized
Bit 6 : '1' - parity error recognized
'0' - no parity error recognized
Bit 5 : '1' - framing error recognized
'0' - no framing error recognized
Bit 4 : '1' - receiver overrun recognized (data loss!)
'0' - no receiver overrun recognized
Bit 3 : '1' - Rx time-out occurred
'0' - no Rx time-out occurred
Bit 2 : '1' - character in the local interrupt buffer
'0' - no character in the local interrupt buffer
Bit 1 : '1' - receiver has set handshake to 'disabled'
'0' - receiver has set handshake to 'enabled'
Bit 0 : '1' - receiver disabled
'0' - receiver enabled

errlog enable/disable Rx-error function, read only.

errlog = $00 - no Rx-error function
errlog = $FF - Rx-error function enabled

errlog is set by the command *receive-errlog*.
errlog is reset by *receive-on* and *receive-off*.

2.2.3 Command Handing-over via the Parameter Channel

Via the parameter channel commands can be handed over as well as parameters of the data buffer. For this purpose, the parameter channel is entered into the Tx server queue and thus being executed synchronously.

The commands 'clear' and 'reset', are already executed before being entered into the queue.

The corresponding command is entered into the location **iocmmd** in the header of the parameter channel.

Already implemented commands:

$0000	**paraxy**
$000C	**clear**
$000D	**reset**
$000E	**reset-Status**
$0050	**receive-Off**
$0051	**receive-On**
$0052	**receive-Errlog**
$FFFF	**sync**

Description of the commands:

paraxy changes interface parameters, as e.g. baud rate, handshake

clear deletes the locally stored RX data;
resets the output queue, changes no interface parameters

reset default initialization of the channel

reset-stat resets the *error* flags in *txstat* and *rxstat*

receive-off switches the receiver off

receive-on switches the receiver on (no ‘end-by-error')

receive-errlog switches the receiver on, enables the 'end-by-error' function

sync entering the parameter channel as an output without data, no data transfer, no change of the interface status.

At heavy duty transmit operation without 'wait for ready' (MODMWA in *iomode*=0) the condition 'output queue full' will easily become true, thus the master must check for 'output queue ready' in the polling mode.
However, after the next transfer the queue is full again. At this condition we recommend to execute a dummy transfer with 'wait for ready' and an activated interrupt mode. Thus after a complete execution of the queue the total memory is available to the master again.

2.3 Description of the Interrupter Channel

2.3.1 Structure of the Interrupter Channel

The task of the interrupter channel is to establish a connection between the VME master program and the local server.

After allocating a data channel and entering the parameters into the header of this channel, the master program must hand over the channel to the local server. For this, the interrupter channel makes available the cells *TCHACH1* to *TCHACHA* and *RCHACH1* to *RCHACHA* in its data buffer.

The master program enters the **board relative** address of the channel to be accessed (D0 in the example mentioned above) into these cells and activates the VME-ISER server by triggering a local interrupt. The VME-ISER server identifies the data channel by the entry in the interrupter channel and thus can work on it.

The interrupter channel makes available an entry each both for transmit and receive operation for each of the 10 interfaces.

The cells *TCHACHx*/*RCHACHx* serve as status cells as well:

If the content of the cell *CHACHx* is unequal to `$00000000.L`, the corresponding data channel has not yet been integrated into the VME-ISER server queue, and no new entry may take place.

As soon as the data channel is integrated into the server management, the entry in the interrupter channel is set to `$00000000.L`. This entry delivers no information about the status of the corresponding channel. The status can only be obtained from the condition of the cell *iosema* in the header of the data channel!

Address Offset HEX	+0		+2		+4		+6		+8		+A		+C		+E	
8000	00	00	**81**	**00**	00	00	00	00	**FF**	**FF**	***'Irch__'***					
8010	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	**80**
8020	00	00	**80**	**80**	00	00	00	00	00	00	00	00	00	00	00	00
8030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
8040	*addr_para1*				*addr_para2*				*addr_para3*				*addr_para4*			
8050	*addr_para5*				*addr_para6*				*addr_para7*				*addr_para8*			
8060	*addr_para9*				*addr_paraA*				00	00	00	00	00	00	00	00
8070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
8080	*TCHACH1*				*TCHACH2*				*TCHACH3*				*TCHACH4*			
8090	*TCHACH5*				*TCHACH6*				*TCHACH7*				*TCHACH8*			
80A0	*TCHACH9*				*TCHACHA*				00	00	00	00	00	00	00	00
80B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
80C0	*RCHACH1*				*RCHACH2*				*RCHACH3*				*RCHACH4*			
80D0	*RCHACH5*				*RCHACH6*				*RCHACH7*				*RCHACH8*			
80E0	*RCHACH9*				*RCHACHA*				00	00	00	00	00	00	00	00
80F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Table 2.3.1: Interrupter Channel

2.3.2 Description of the Interrupter Channel Cells

addr_para1... Start addresses of the parameter channels 1 to 10
adr_paraA

TCHACH1...
TCHACHA Entries for the **Tx server**:

Cell	Offset [HEX] relative to *iodata*	Entry Channel for Tx Server
TCHACH1	00	1
TCHACH2	04	2
TCHACH3	08	3
TCHACH4	0C	4
TCHACH5	10	5
TCHACH6	14	6
TCHACH7	18	7
TCHACH8	1C	8
TCHACH9	20	9
TCHACHA	24	10

Table 2.3.2: Entries for the Tx server

Triggering of the local VME-ISER-Tx-Irq's:

To activate the VME-ISER Tx server task, which executes the entries in the interrupter channel, an access to the local IRQ trigger address must take place.
This access must ensue as 'write word' to the **board relative** address:

tirtrig = $080002

RCHACH1...
RCHACHA Entries for the **Rx server**:

Cell	Offset [HEX] relative to *iodata*	Entry Channel for Rx Server
RCHACH1	40	1
RCHACH2	44	2
RCHACH3	48	3
RCHACH4	4C	4
RCHACH5	50	5
RCHACH6	54	6
RCHACH7	58	7
RCHACH8	5C	8
RCHACH9	60	9
RCHACHA	64	10

Table 2.3.3: Entries for the Rx server

Triggering the local VME-ISER-Tx-Irq's:

To activate the VME-ISER Rx server task, which executes the entries in the interrupter channel, an access to the local IRQ trigger address must take place.
This access must ensue as 'write word' to the **board relative** address

rirtrig = $080006.

3. The local VME-ISER Server

3.1 Functional Description of the local VME-ISER Server

The local VME-ISER server manages all channels, which have been handed over from the VME master program to the VME-ISER. The server distinguishes basically between **input and output** channels. The execution of a parameter channel is a special form of an output channel.

3.1.1 Output Channels

The VME-ISER server contains a local execution queue for each interface. As a default these queues have a depth of 32 entries. An output data channel linked in via the interrupter channel will be entered into the queue and the **Tx server**, responsible for the interface, obtains the particular channel from the queue and releases the entry again after the complete execution.

A run-over of the queue is prevented by the handshake with the cells *TCHACHx*: if the queue is full, the entry of the corresponding data channel is certainly accepted, but the cell *TCHACHx* will not yet be released again. This will only happen, if space for at least one more entry is available in the queue.

If the TX server recognizes the actual output channel as a parameter channel, no output will occur, but the command **iocmmd** will be executed.

3.1.2 Input Channels

An interrupt buffer is allocated to each of the 10 serial interfaces as a default. The user has no direct access to this buffer.

If data are received via the interface, and there is no input buffer available to the input server, then the incoming data will be temporarily stored in the interrupt buffer.

As long as there are still data in the interrupt buffer, an input channel linked in by the VME master will be filled with these data, otherwise incoming data are directly transferred into the input channel.

Exceptions:

- if an input channel with *iomode*=`$xx08` is processed, all data up to now received in the interrupt buffer are deleted, and only data received from now on will be handed over at the next READ instruction.

- If *iofnam* is set to ASCII 'SCAN', data from the interrupt buffer will be handed over until reaching the indicated end condition. If the interrupt buffer is clear, the end condition will also be set.

- If *iofnam* is set to ASCII ‘PROT’, the registered protocol will be executed.

As a default the interrupt buffer has a length of 1 kbyte. The receive handshake is managed corresponding to the free space of the interrupt buffer:

If the interface is equipped with a handshake, at a remaining space of about 10% the handshake is disabled.
If the free space is about 70% again, the handshake will be enabled again.

3.1.3 Interrupt Operation

If the user needs a VME interrupt from the VME-ISER after completing an instruction (e.g. input channel filled, or output channel transferred with *MODMWA* = '1' in *iomode*), then the desired VME interrupt level, as well as the interrupt vector must be entered into the cells *ioilev* and *ioivec* of the corresponding data channel. The VME-ISER then generates the specified interrupt.
If no interrupt generation is desired, *iolev* must be set to 0.
In his interrupt routine the user must confirm the interrupt. The interrupt confirmation is done as follows:

The 2 LSB of the interrupt vector determine the bit position in the interrupt acknowledge register. This bit must be set to '1' as an acknowledge. The **board relative** address of this register ***iack*** is $08601B.

e.g.:
```
---   Interrupt-Entry  ---
   MOVE.B  #ioivec,D0          ;actual interrupt vector
   ANDI.B  #$03,D0             ;Masking bit 2 to 7
   BSET    D0,iack+iserbase    ;Set bits on VME-ISER
--- further interrupt routine --
```

Setting the IACK bit should happen as soon as possible, because on the VME-ISER the generation of a new IRQ is prevented as long as the actual interrupt was not confirmed!!

3.1.4 Time-Out

Optionally it is possible to abort transmit and receive instructions after a preset time *T-Out*. Time setting is done via the channel parameter *iotout*, or via the parameters *rtime0*, *rtime1* and *ttime* in the parameter channel.
The value in *iotout* corresponds to the channel being executed, while *rtime0*, *rtime1* and *ttime* refer to the interface in general.
The content of *iotout* overdrives the content in the parameter channel.

iotout If bit 7 of *iotout* equals to 0, then a time-out via <*iotout*> is disabled.
If bit 7 equals to 1, then the value of the remaining 7 bits indicates the time-out time in multiples of 10 msec.
e.g.:
iotout = $0x - no time-out
iotout = $85 - time-out after 50 msec
iotout = $FF - time-out after 1.2 sec

It is possible to set a global time-out for all interfaces via the parameter channel, which can be different for transmit and receive operation.
The range of values is 0 . . . . 32767, the unit is 1 msec.
If *rtime0*= 0, or *ttime*= 0, then the corresponding time-out function is disabled!

ttime	time-out for transmit operation
rtime0	time-out for receive operation for the first character
rtime1	time-out for receive operation for any further character

The time-out function is retriggerable, i.e. if a transmit or receive operation takes place, the corresponding counter will be reset. The chronological interval of these operations is variable (FIFO operation) and corresponds to the duration of at least one, but as a maximum of 8 character times.
(e.g. 1200 Baud: 1 char.time . (1+8+1)/1200 = 8.3 msec
8 char.times . 66.6 msec, i.e. a time-out value
of less than 67 msec cannot be recommended!)

Moreover, in the receive operation it is distinguished between ‘first’ time-out and ‘character-to-character’ time-out, i.e. the time between instruction input and first character arrival may be longer than the character-to-character time while the active transfer.

Actions when a time-out occurs:

If a time-out occurs at a transfer, the following actions happen as a principle:

1. in the corresponding channel the time-out mark is set:
   $8007 --> *iorecl*

2. in the parameter channel the time-out bit in *rxstat*, or in *txstat* is set.

The reset of these bits is done via the command **reset-stat** in the parameter channel or at a channel reinitialization. The bit is **not** reset at a successful input or output!

The channel being worked on is released again, i.e. at a transmit channel without 'wait' the channel will be 'scrapped'. The semaphore *iosema* is reset and the next transmit channel is obtained from the queue.

At a transmit operation with 'wait', or at a receive channel the master is informed correspondingly. The channel status is set to 'ready' and, if required, an interrupt is generated.

3.1.5 Receive Error Mode

Errors occurring in the Rx mode are recorded in *rxstat*.
An Rx status reset is performed by the commands

reset-stat, **reset** or **receive-errlog**.

Detectable errors are break, parity, framing and overrun errors.

If an evaluation of these errors is desired, then the receiver error mode must be activated by the command **receive-errlog**.

If one of the above-mentioned errors occurs in the active mode, and no receive instruction is effective, all characters received in the interrupt buffer will be deleted. If an Rx instruction is effective, the instruction is aborted and an error code is returned via *iorecl*.

If several errors occur simultaneously, following priority will be obeyed: break/parity error/framing error/overrun error.

Error codes in *iorecl*:

`$8007` - time-out
`$801E` - framing error
`$801F` - overrun error
`$8020` - parity error
`$8046` - break detected

The error condition time-out is independent of the condition *errlog*, and is released only by the time-out cells described before.

3.2 Examples for the VME-ISER Server

3.2.1 Example: Initialization of the VMEbus Master

It is recommended to let the initialization routine of the master determine the following addresses once and store them in master-local cells:

CRDADR -- VMEbus base address of the VME-ISER
TxBUFF -- VME-ISER relative address of the Tx channels 1 to 10
RxBUFF -- VME-ISER relative address of the Rx channels 1 to 10
PARAn -- VME-ISER relative address of the parameter cannel. 1 to 10
IRCH -- VME-ISER relative address of the interrupter channel data buffer (*iobuff*(IRCH))
IACK -- interrupt acknowledge address absolute
TIRTRIG -- transmit interrupt trigger address absolute
RIRTRIG -- receive interrupt trigger address absolute

The master should scan the VME-ISER channels, starting with the address of **ANCHOR** and either check for the corresponding ASCII string (*TBUFxy*, *RBUFxy*, *PARAxy* and *Irch*) or determine the channel via the cells *iotyp* and *ioldn*. As next-pointer *iofor* has to be used.

3.2.2 Example: Data Output to Interface 2 without IRQ

```
TCHACH1    EQU    (1-1)*4              ;offset server 1
TCHACH2    EQU    (2-1)*4              ;offset server 2
..
TCHACH9    EQU    (9-1)*4              ;offset server 9
TCHACHA    EQU    (10-1)*4             ;offset server A


           MOVEA.L CRDADR,A0           ;base address
           MOVE.L  TXBUF2,D0           ;first channel
           BSR     srchbf              ;search for free channel
                                        (see above)
           BNE     wait                ;no channel free, wait !?
*          Now A0 contains the absolute address of the actual
*          channel, D0 contains the board relative address
           MOVEA.L   iobuff(A0),A1     ;rel. address data buffer
           ADDA.L    CRDADR,A1         ;absolute address
           MOVE.W    #anzdata,D1       ;number of data bytes
           MOVE.W    D1,iorecl(A0)     ;enter into header
           SUBQ      #1,D1             ;because of DBxx
           MOVEA.L   source,A2         ;pointer to transmit data
loop       MOVE.B    (A2)+,(A1)+       ;transfer to VME-ISER
           DBF       D1,loop           ;
           MOVE.W    #0,ioilev(A0)     ;ioilev,ioivec == $0
           MOVE.L    #0,iofnam(A0)     ;clear fname
           MOVE.W    #$4700,iomode(A0) ;output, no wait
*          activate VME-ISER server
           MOVEA.L   IRCH,A2           ;pointer to data interrup.
           ADDA.L    CRDADR,A2         ;absolute
           TST.L     TCHACH2(A2)       ;entry free ?
           BNE       wait              ;No, wait ?
           MOVE.L    D0,TCHACH2(A2)    ;enter relative channel address
           MOVE.W    D0,TIRTRIG        ;write 'any' as a trigger
* ----     ready     ---
```

3.2.3 Example: Data Input from Interface 8

```
RCHACH1    EQU    (1-1)*4+$40          ;offset server 1
RCHACH2    EQU    (2-1)*4+$40          ;offset server 2
..
RCHACH9    EQU    (9-1)*4+$40          ;offset server 9
RCHACHA    EQU    (10-1)*4+$40         ;offset server 10


           MOVEA.L CRDADR,A0           ;base address
           MOVE.L  RXBUF,D0            ;first channel
           TAS     iosema(A0,D0.L)     ;search for free channel
*                                       (see above)
           BNE     wait                ;no channel free, wait !?
           LEA     0(A0,D0.L),A0
*          Now A0 contains the absolute address of the actual
*          channel, D0 contains the board relative address
           MOVE.W  #anzdata,D1         ;maximum number of the
*                                       data bytes to be read
           MOVE.W  D1,iorecl(A0)       ;enter into header
           MOVE.B  #05,ioilev(A0)      ;IRQ level = 5
           MOVE.B  #$60,ioivec(A0)     ;IRQ vector =$60
           MOVE.W  #$2700,iomode(A0)   ;input, end at <cr>
           MOVE.L  #0,iofnam(A0)       ;normal input
*          activate VME-ISER server
           MOVEA.L IRCH,A2             ;pointer to data interrup.
           ADDA.L  CRDADR,A2           ;absolute
           TST.L   RCHACH8(A2)         ;entry free ?
           BNE     wait                ;no, wait ?
           MOVE.L  D0,RCHACH2(A2)      ;enter relative channel address
           MOVE.W  D0,RIRTRIG          ;write 'any' as a trigger
*
* ----     wait until occurring of the special IRQ
           MOVE.W  iorecl(A0),D1       ;number of received data
           BEQ     exit                ;no data received
           BMI     error
           SUBQ    #1,D1               ;because of DBxx
           MOVEA.L destin,A2           ;destination of the data
           MOVEA.L iobuff(A0),A1       ;source of the data, relative
           ADDA.L  CRDADR,A1           ;address absolute
loop1      MOVE.B  (A1)+,(A2)+         ;transfer data bytes
           DBF     D1,loop1            ;
           MOVE.B  =0,iosema(A0)       ;release channel !!
*  ----    ready   --
*
error      ANDI.W  =$7FFF,D1           ;mask error number
           .
           . (error routine)
           .
```

3.2.4 Example: Setting the Parameter of Interface 1

```
TCHACH1    EQU    (1-1)*4              ;offset server 1
TCHACH2    EQU    (2-1)*4              ;offset server 2
..
TCHACHA    EQU    (10-1)*4             ;offset server 10
txbs       EQU    0                    ;desired value Tx_Baud
rxbs       EQU    txbs+1
chrls      EQU    rxbs+1
stpls      EQU    chrls+1
parts      EQU         stpls+1
hnds       EQU    parts+1
txb        EQU    $40                  ;actual value Tx_Baud
rxb        EQU    txb+1
chrl       EQU    rxb+1
stpl       EQU    chrl+1
part       EQU    stpl+1
hnd        EQU    part+1

           MOVEA.L CRDADR,A0           ;base address
           MOVE.L  PARA1,D0            ;parameter channel, relative
           ADDA.L  D0,A0               ;absolute address
           MOVEA.L iobuff(A0),A1       ;data range parameters
           ADDA.L  CRDADR,A1           ;absolute address

*          e.g.:
*          set tx baud rate to 300 Baud
*          set rx baud rate to 600 Baud
*          set handshake to XON/XOFF
           MOVE.B  #7,txbs(A1)         ;tx Baud = 300
           MOVE.B  #6,rxbs(A1)         ;rx Baud = 600
           MOVE.B  #1,hnds(A1)         ;XON/XOFF handshake
*          All other parameters remain unchanged
           MOVE.W  #$4700,iomode(A1);output mode
           MOVE.W  #0,ioilev(A1)       ;no IRQ
           MOVE.W  #0,iocmmd(A1)       ;mode: Init parameter
*          enter parameter channel into server queue
           MOVEA.L IRCH,A2             ;pointer to data interrup.
           ADDA.L  CRDADR,A2           ;absolute
           TST.L   TCHACH1(A2)         ;entry free ?
           BNE     wait                ;no, wait ?
           MOVE.L  D0,TCHACH1(A2)      ;enter relative channel address
           MOVE.W  D0,TIRTRIG          ;write 'any' as a trigger
*  ----    ready   ----
```

3.3 User Protocols

3.3.1 Function Description

The user has got the possibility to implement an individual Rx-protocol or Rx-filter for each channel. In order to do this the protocol program has to be loaded in an available RAM-area of the VME-ISER (such as $20000... $3FFFF) and the entry address of the local user program has to be made available to the local ISER server. This can be achieved by specifying the entry address of the respective channel in cell *ioentr* in the parameter channel.

If the VME master now requests an Rx-element via *iofnam* = PROT, the received characters are buffered in the interrupt buffer, followed by the execution of the specified protocol which can check the buffered chain of characters and possibly transmit them to the requested channel.

If *iofnam* of the requested channel unequals PROT, the data is transferred normally by means of the standard VME-ISER server.
If *iofnam* of the requested channel equals PROT, and if the protocol entry *ioentr* is not available, the Rx-request will be ignored.

It is very important to ensure that the basic configuration of the channel via the parameter channel does not cause conflicts with the requested protocol (such as a software handshake in binary protocols)!

3.3.2 Conditions for the Use of User-Specific Rx-Protocols/Filters

- the application program has to be installed in a free memory range between $20000 and $3FFFE
- the entry address of the server routine has to be specified in the respective parameter channel in cell *IOENTR*
- the entry address has to be even
- the last four bytes before the entry address have to include the ASCII-ID 'PROT'
- Re-entry window, freely relocatable 68000-Code
  no commands for 68020/30/40!
- no software traps
- restrictions in the use of registers:
  Register A1 contains the pointer to the variables of the respective channel (such as *irwp, ceaddr,...*).
  Register A3 contains the return address. In register D0 the status of the protocol is returned:
  '0' - Prot. not yet finished
  > 0 - number of bytes
  < 0 - e.g. number of bytes + bit 15 set: CRC-error
  Data registers A2, A4, D1, D2, D4 can be used. A1 and A3 must not be changed!

The protocol is entered in supervisory mode on interrupt level 5 or interrupt level 7.

3.3.3 Register and Structure Declarations

Register	
A1.L	pointer to structure *irbuf*
A3.L	return address
A2.L/A4.L	free
D0.L/D1.L/D2.L/D4.L	free

When returning from the protocol via 'JMP(A3)' D0.W has to be supplied with the returned value and the according flags have to be set in the status register:

Returned values in D0.W:

D0	Flags	
'0'	'eq'	Protocol has not been finished yet, no further action of the ISER server.
'$0001','m'	'ne','pl'	Protocol has been finished without errors, *m* characters have been transmitted to the Rx-buffer: The VME-ISER server returns the Rx-buffer to the VME-master.
'$8000','m+$8000'	'ne','mi'	Protocol has been finished with errors, *m* characters have been transmitted to the Rx-buffer: The VME-ISER server returns the Rx-buffer to the VME-master.

Data Structure *irbuf* (Interrupt Buffer)

Each VME-ISER channel has got an *irbuf* structure via which the Tx- and Rx-transfers are processed. Into this structure the received data, for instance, is filed. It consists mainly of four parts:

- pointer and counter for Tx-operation
- queue for Tx-operation (32 entries)
- pointer and counter for Rx-operation
- FIFO for Rx-operation (1024 bytes)

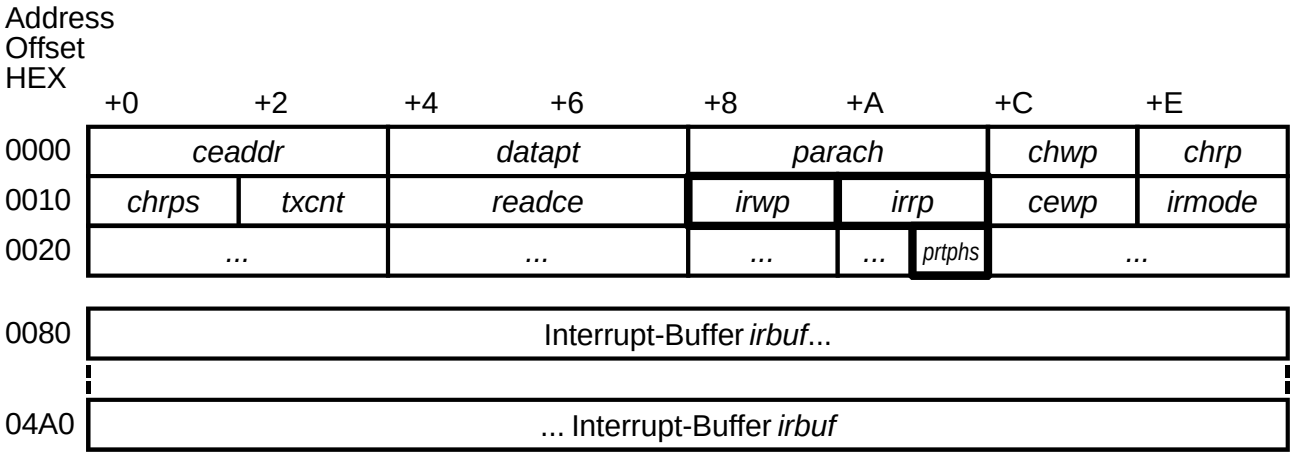


Table 3.3.1: Relevant cells of the interrupt buffer

Usually, the following structure elements of the **interrupt buffer** satisfy the Rx-protocol:

Name	Offset [HEX]	Organisation	Meaning
readce	14	longword	absolute address of the waiting Rx-buffer (*iobuff*)
irwp	18	word	current write pointer in the data range *irbuf0* (*can* be set by the protocol to synchronise)
irrp	1A	word	current read pointer in the data range *irbuf0* (*must* be managed by the protocol)
prtphs	2B	byte	flags to control the protocol
irbuf	40		interrupt buffer, length: `$400`

Table 3.3.2: Relevant structure elements of the interrupt buffer

Note:
Apart from cells *irrp*, *prtphs* and possibly *irwp* all other cells are read-only for the application program!

Furthermore, a pointer is required from the **data channel** (structure *iobuff*):

Name	Offset [HEX]	Organisation	Meaning
iobuff	20 *)	longword	pointer to data range

*) Offset in data channel!

Table 3.3.3: Pointer to data range

3.3.4 Protocol Embedding for Rx-Operation

If characters are received, they are read-out of the controller on interrupt level, are possibly checked for signs of software handshake or ‘end’ signs, and filed in the Rx-FIFO. Then, if required by the Rx-buffer, the user protocol is executed. This can now check the characters while knowing the current write pointer *irwp* and the (self-administered) read pointer *irrp*. If the protocol requirements are not met, the returned parameter ‘0’ is transmitted and the protocol is activated again when the following characters are received.

If the protocol requirements have been met, the application program will initiate the transfer of characters into the Rx-buffer: The pointer to the waiting Rx-buffer is in cell *readce*, and is of structure type *iobuff*. In cell *iobuff* of this structure is the initial address of the data range into which the characters are to be transferred.

After all characters have been transferred, the number of valid bytes is transferred in D0; the MSB can be used as a flag for a faulty protocol. According to the configuration, the VME-ISER server then returns the Rx-buffer to the VME-master.
Register A1 is the basic address for the current structure *irbuf* and must not be changed during the protocol!

Please make sure that the time for the protocol processing is optimized on server level, because no further characters can be handled during this time (data loss!)!

Example:

```
          DC.B     'PROT'
entry:    LEA      irbuf0(A1),A2      ; A2: pointer to Rx-data range
          MOVE.W   irrp(A1),D1        ; last read pointer
          MOVE.B   0(A2,D1.W),D0      ; character from Rx-buffer
          CMPI.B   =char,D0           ; checking the character
          BNE.S    exit               ; not OK
          ADD.W    =len,D1            ; next read pointer
          MOVE.L   D1,irrp(A1)
transfer  LEA      0(A2,D1.W),A2      ; pointer to character chain
          MOVEA.L  readce(A1),A4      ; pointer to Rx-buffer 'iobuff'
          MOVEA.L  iobuff(A4),A4      ; pointer to Rx-data range
          MOVE.W   =len-1,D2          ; transfer length
tloop     MOVE.B   (A2)+,(A4)+        ; transfer character chain
          DBF      D2,tloop           ;
          MOVE.W   =len,D0            ; returned value
          JMP      (A3)               ; to VME-ISER server
exit      MOVEQ    =0,D0              ; flag: not ready yet
          JMP      (A3)               ;
```

When accessing the data range in the interrupt buffer, you have to remember that it is a FIFO with 1 k byte length, which means that all pointers have to be treated Modulo $3FF!

Example for Configuration (esn-stx/etx-Protocol):

For this protocol the following configuration is advisable:

- *iorecl* = $0018
- *iofnam* = 'PROT'
- *iomode* = $8700
- *ioivev, ioilev* = $00 , $00 - no interrupt, or
- *ioivec, ioilev* = vector, level - user-defined IRQ

For the group configuration via the parameter channel:

txbs = $13 (115200 baud)
rxbs = $13 (115200 baud)
chrls = $00 (8 bits/char)
stpls = $00 (1 stop bit)
parts = $00 (no parity)
hnds = $03 (no handshake)
rtime0s/rtime1s = $0000 or time-out in msec ($ 3 !)

Index

V