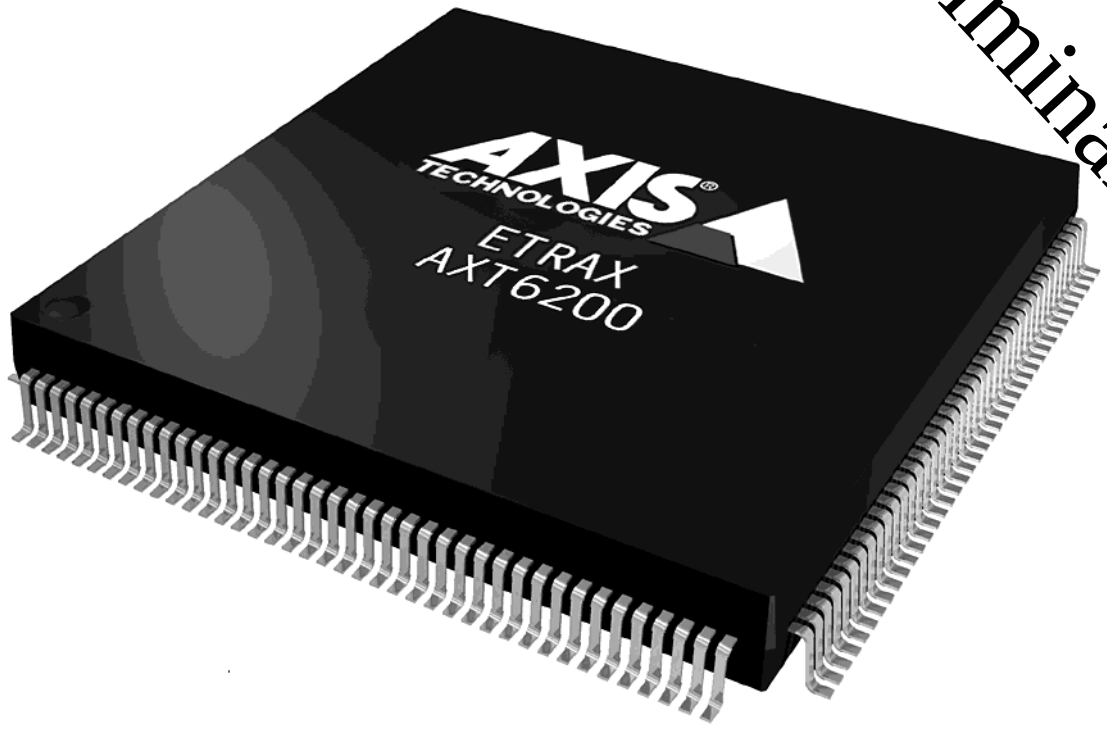
Preliminary



AXT6100 & AXT6200
ETRAX Designer's Reference

Highly Integrated Token Ring
and Ethernet Controller with RISC
CPU and Multiple I/O Ports

AXIS®
TECHNOLOGIES

Axis – the intelligent choice

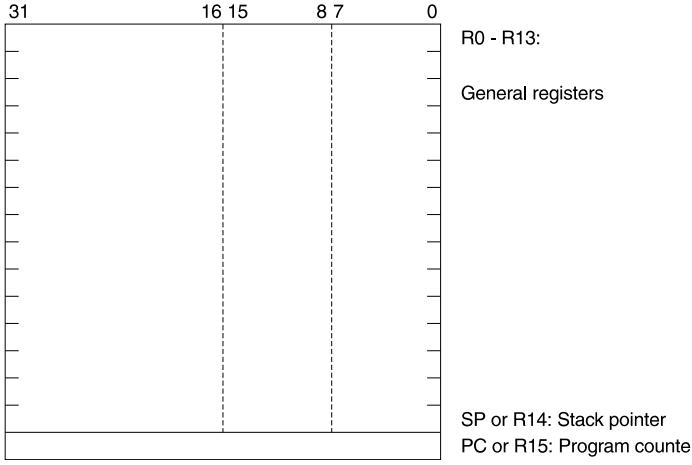
Table of contents

1 RISC CPU

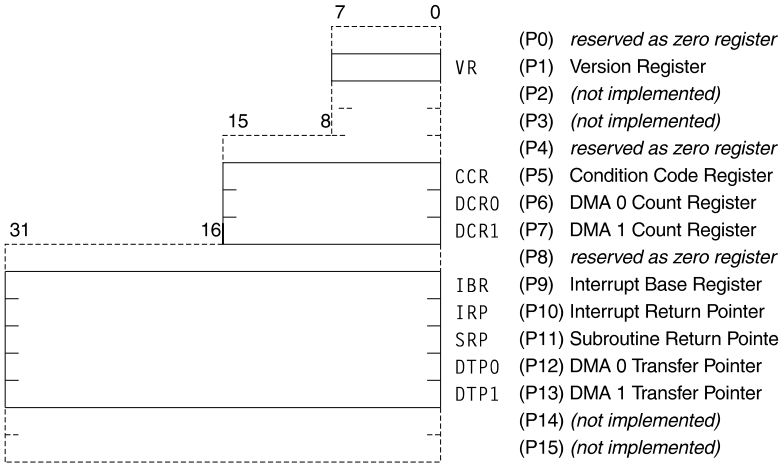
This block is a 32-bit RISC CPU with a 16-bit instruction width. The CPU complies with the Axis proprietary CRIS architecture. It runs at a bus frequency of 16 MHz, giving a peak performance of 16 MIPS. The CPU contains two DMA channels with 32-bit address pointers and 16-bit counters. A summary of the CRIS architecture is given below. The CRIS CPU architecture is described in more detail in 'CRIS Programmers Reference Manual'.

1.1 REGISTERS

The processor contains 15 32-bit General Registers (R0 - R14), a 32-bit Program Counter and 9 Special Registers.



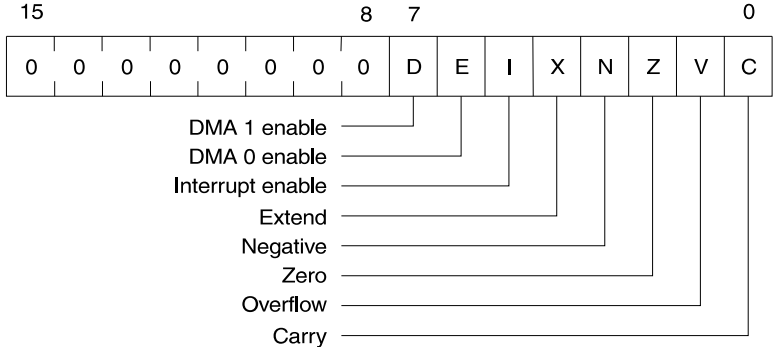
1-1 General Registers



1-2 Special Registers

1.2 FLAGS AND CONDITION CODES

The condition code register (CCR) contains eight different flags. The eight remaining bits of the CCR are always zero.



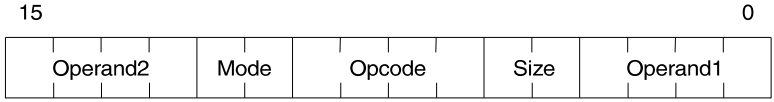
1-3 Condition code register

The flags can be tested using one of the 16 condition codes specified below:

Code	Alt	Condition	Encoding	Boolean function
CC	HS	Carry Clear	0000	$\overline{C}$
CS	LO	Carry Set	0001	C
NE		Not Equal	0010	$\overline{Z}$
EQ		Equal	0011	Z
VC		Overflow Clear	0100	$\overline{V}$
VS		Overflow Set	0101	V
PL		Plus	0110	$\overline{N}$
MI		Minus	0111	N
LS		Low or Same	1000	C + Z
HI		High	1001	$\overline{C} * \overline{Z}$
GE		Greater or Equal	1010	$N * V + \overline{N} * \overline{V}$
LT		Less Than	1011	$N * \overline{V} + \overline{N} * V$
GT		Greater Than	1100	$N * V * \overline{Z} + \overline{N} * \overline{V} * \overline{Z}$
LE		Less or Equal	1101	$Z + N * \overline{V} + \overline{N} * V$
A		Always True	1110	1
EXT		External Pin	1111	external input

1.3 INSTRUCTION FORMAT

The basic instruction word is 16 bits long. Instructions must be Word (16 bit) aligned. The most common instructions follow the same general instruction format:



1-4 General instruction format

1.3.1 Addressing Modes

The CRIS CPU has four basic addressing modes. These modes are encoded in the mode field of the instruction word. The basic addressing modes are:

- Quick immediate mode
- Register mode
- Indirect mode
- Autoincrement mode (with immediate mode as a special case)

More complex addressing modes can be achieved by combining the basic instruction word with an addressing mode prefix word. The complex addressing modes are:

- Indexed
- Indexed with assign
- Offset
- Offset with assign
- Double indirect
- Absolute

The addressing modes of the CRIS CPU are described below.

Assembler syntax	Addressing mode
i , j	Quick immediate
Rn	Register
Pn	Special register
[Rn]	Indirect
[Rn+]	Post increment
x , u	Byte immediate
xx , uu	Word immediate
xxxx , uuuu	Dword immediate
[Rn+Rm.m]	Indexed
[Rp=Rn+Rm.m]	Indexed with assign
[Rn+[Rm].m]	Indirect offset
[Rn+[Rm+].m]	Autoincrement offset
[Rn+x]	Immediate byte offset
[Rn+xx]	Immediate word offset
[Rn+xxxx]	Immediate dword offset

Assembler syntax	Addressing mode
[Rp=Rn+[Rm].m]	Indirect offset with assign
[Rp=Rn+[Rm+].m]	Autoincrement offset with assign
[Rp=Rn+x]	Immediate byte offset with assign
[Rp=Rn+xx]	Immediate word offset with assign
[Rp=Rn+xxxx]	Immediate dword offset with assign
[[Rn]]	Double indirect
[[Rn+]]	Double indirect with autoincrement
[uuuu]	Absolute

1.3.2 Data Transfers

The data transfer instructions of the CRIS CPU are shown in the table below. The two predefined assembler macros POP and PUSH are also shown in the table.

Instruction	Flag operation								Description
	D	E	I	X	N	Z	V	C	
CLEAR.m d	–	–	–	0	–	–	–	–	Clear destination operand
MOVE.m s,Rd	–	–	–	0	*	*	0	0	Move from source to general register
MOVE.m Rs,di	–	–	–	0	–	–	–	–	Move from general register to memory
MOVE s,Pd	–	–	–	0	–	–	–	–	Move from source to special register
MOVE Ps,d	–	–	–	0	–	–	–	–	Move from special register to destination
MOVEM Rs,di	–	–	–	0	–	–	–	–	Move multiple registers to memory
MOVEM si,Rd	–	–	–	0	–	–	–	–	Move from memory to multiple registers
MOVEQ i,Rd	–	–	–	0	*	*	0	0	Move 6-bit signed immediate
MOVS.z s,Rd	–	–	–	0	*	*	0	0	Move with sign extend
MOVU.z s,Rd	–	–	–	0	0	*	0	0	Move with zero extend
POP Rd	–	–	–	0	*	*	0	0	Pop register from stack
POP Pd	–	–	–	0	–	–	–	–	Pop special register from stack
PUSH Rs	–	–	–	0	–	–	–	–	Push register onto stack
PUSH Ps	–	–	–	0	–	–	–	–	Push special register onto stack

1.3.3 Arithmetic Instructions

The arithmetic instructions of the CRIS CPU are described in the table below.

Instruction	Flag operation								Description
	D	E	I	X	N	Z	V	C	
ABS Rs,Rd	–	–	–	0	*	*	0	0	Absolute value
ADD.m s,Rd	–	–	–	0	*	*	*	*	Add source to destination register
ADDI Rs.m,Rd	–	–	–	0	–	–	–	–	Add scaled index to base
ADDQ j,Rs	–	–	–	0	*	*	*	*	Add 6-bit unsigned immediate
ADDS.z s,Rd	–	–	–	0	*	*	*	*	Add sign extended source to register
ADDU.z s,Rd	–	–	–	0	*	*	*	*	Add zero extended source to register
BOUND.m s,Rd	–	–	–	0	*	*	0	0	Adjust table index (unsigned min)
CMP.m s,Rd	–	–	–	0	*	*	*	*	Compare source to register
CMPQ i,Rd	–	–	–	0	*	*	*	*	Compare with 6-bit signed immediate
CMPS.z si,Rd	–	–	–	0	*	*	*	*	Compare w. sign extended source
CMPU.z si,Rd	–	–	–	0	*	*	*	*	Compare w. zero extended source
DSTEP Rs,Rd	–	–	–	0	*	*	0	0	Divide step
MSTEP Rs,Rd	–	–	–	0	*	*	0	0	Multiply step
NEG.m Rs,Rd	–	–	–	0	*	*	*	*	Negate (2's complement)

Instruction	Flag operation D E I X N Z V C	Description
SUB.m s,Rd	– – – 0 * * * *	Subtract source from register
SUBQ j,Rd	– – – 0 * * * *	Subtract 6-bit unsigned immediate
SUBS.z s,Rd	– – – 0 * * * *	Subtract w. sign extended source
SUBU.z s,Rd	– – – 0 * * * *	Subtract w. zero extended source
TEST.m s	– – – 0 * * 0 0	Compare operand with 0

1.3.4 Logical Instructions

The logical instructions of the CRIS CPU are described in the table below.

Instruction	Flag operation D E I X N Z V C	Description
AND.m s,Rd	– – – 0 * * 0 0	Bitwise logical AND
ANDQ i,Rd	– – – 0 * * 0 0	AND w. 6-bit signed immediate
NOT Rd	– – – 0 * * 0 0	Logical NOT (1's complement)
OR.m s,Rd	– – – 0 * * 0 0	Bitwise logical OR
ORQ i,Rd	– – – 0 * * 0 0	OR w. 6-bit signed immediate
XOR Rs,Rd	– – – 0 * * 0 0	Bitwise Exclusive OR

1.3.5 Shift Instructions

The shift instructions of the CRIS CPU are shown in the table below. When the shift count is contained in a register, the 6 least significant bits of the register are used as an unsigned shift count.

Instruction	Flag operation D E I X N Z V C	Description
ASR.m Rs,Rd	– – – 0 * * 0 0	Right shift Rd w. sign fill
ASRQ c,Rd	– – – 0 * * 0 0	Right shift Rd w. sign fill
LSL.m Rs,Rd	– – – 0 * * 0 0	Left shift Rd w. zero fill
LSLQ c,Rd	– – – 0 * * 0 0	Left shift Rd w. zero fill
LSR.m Rs,Rd	– – – 0 * * 0 0	Right shift Rd w. zero fill
LSRQ c,Rd	– – – 0 * * 0 0	Right shift Rd w. zero fill

1.3.6 Bit Test Instructions

The bit test instructions of the CRIS CPU are shown in the table below. These instructions set the N flag according to the selected bit in the destination register. The Z flag is set if the selected bit and all bits to the right of it are zero.I

Instruction	Flag operation D E I X N Z V C	Description
BTST Rs,Rd	– – – 0 * * 0 0	Test bit Rs in register Rd
BTSTQ c,Rd	– – – 0 * * 0 0	Test bit c in register Rd

1.3.7 Condition Code Manipulation Instructions

The condition code manipulation instructions of the CRIS CPU are shown in the table below. The predefined assembler macros EI, DI and AX are also shown.

Instruction	Flag operation								Description
	D	E	I	X	N	Z	V	C	
AX	–	–	–	1	–	–	–	–	Arithmetic extend (SETF X)
CLEARF <list>	*	*	*	0	*	*	*	*	Clear flags in list
DI	–	–	0	0	–	–	–	–	Disable interrupts. (CLEARF I)
EI	–	–	1	0	–	–	–	–	Enable interrupts. (SETF I)
Scc Rd	–	–	–	0	–	–	–	–	Set register according to cc
SETF <list>	*	*	*	*	*	*	*	*	Set flags in list

1.3.8 Jump and Branch Instructions

The jump and branch instructions of the CRIS CPU are shown in the table below. The predefined assembler macros RET and RETI are also shown. Note that the Bcc, RET and RETI instructions have delayed effect.

Instruction	Flag operation								Description
	D	E	I	X	N	Z	V	C	
Bcc o	–	–	–	0	–	–	–	–	Conditional relative branch
Bcc xx	–	–	–	0	–	–	–	–	Branch w. 16-bit offset
JUMP s	–	–	–	0	–	–	–	–	Jump
JSR s	–	–	–	0	–	–	–	–	Jump to subroutine
JIR s	–	–	–	0	–	–	–	–	Jump to interrupt routine
RET	–	–	–	0	–	–	–	–	Return from subroutine
RETI	–	–	–	0	–	–	–	–	Return from interrupt

1.3.9 No Operation Instruction

Apart from what is described above, the CRIS CPU also has a no operation instruction, NOP.

Instruction	Flag operation								Description
	D	E	I	X	N	Z	V	C	
NOP	–	–	–	0	–	–	–	–	No operation

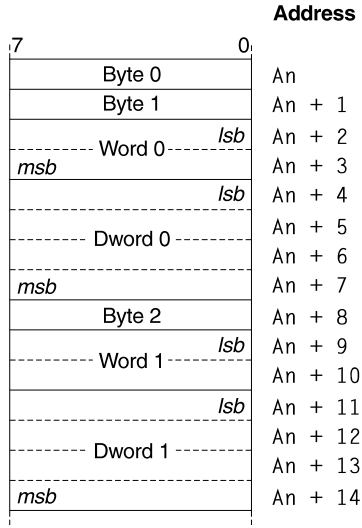
1.4 DATA ORGANIZATION IN MEMORY

The data types supported by CRIS are:

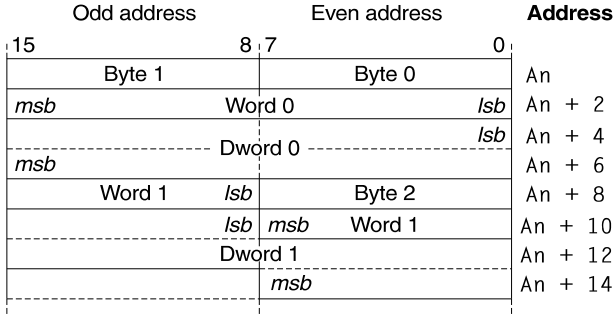
- Byte (8-bit integer)
- Word (16-bit integer)
- Dword (32-bit integer or address)

Each address location contains one byte of data. Word and Dword data is stored in memory with the least significant byte at the lowest address ("little endian").

The CRIS CPU can operate with an 8-bit or a 16-bit wide data bus. The figures below show examples of data organization with an 8-bit bus and a 16-bit bus:



1-5 Data organisation with an 8-bit bus



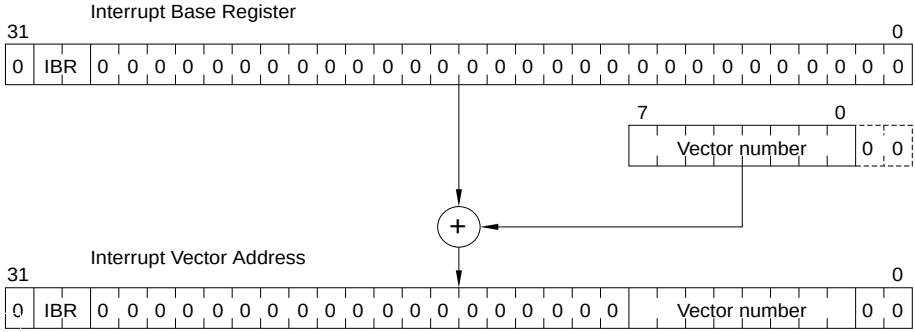
1-6 Data organisation with a 16-bit bus

1.5 INTERRUPTS

The CRIS CPU uses vectored interrupts. (See also section 6, page 52) The interrupt acknowledge sequence consists of the following steps:

1. Perform an INTA cycle, where the 8-bit vector number is read from the bus.
2. Store the contents of PC to the Interrupt Return Pointer (IRP). Note that the return address is not automatically pushed on the stack.
3. Read the interrupt vector from the address [IBR + <vector number> * 4].
4. Start the execution at the address pointed to by the interrupt vector.

Only bit 29 and 30 of the Interrupt Base Register (IBR) are actually implemented. The remaining bits are always zero.



1.6 DMA

The CRIS CPU contains two DMA channels. Each channel has a 32-bit **DMA Transfer Pointer** (**DTP**) , a 16-bit **DMA Count Register** (**DCR**) and a DMA enable flag (**D or E**). The connection of each channel to a physical I/O channel is described in section 1.6.2 below.

To start a DMA transfer, the DTP of the channel is loaded with the start address of the data block to be transferred, and the DCR of the channel is loaded with the number of transfers. (Loading the DCR with 0 will give 65536 transfers). Then the DMA enable flag of the channel is set with the SETF instruction.

For each transfer, the DTP is incremented with 1 (byte transfer) or 2 (word transfer), and the DCR is decremented by 1. When the DCR counts down to 0, the DMA enable flag is set to 0 and the transfers stop.

The DMA can be stopped at any time by clearing the DMA enable flag. Note that the SETF and CLEARF instructions are the only instructions that will affect the D and E flags. When CCR is updated using the MOVE instruction, the D and E flags are left unchanged.

1.6.1 DMA configuration

The DMA configuration block is responsible for connecting the two DMA channels of the CPU with the appropriate internal or external I/O device.

The configuration is controlled by the **R_DMA_CONFIG** register, see page 62.

DMA channel 0 is configurable for:

- parallel in (default)
- parallel out port 1
- parallel out port 2

DMA channel 1 is configurable for:

- parallel out port 1
- serial port out
- External I/O

DMA channel 1 has priority over channel 0.

1.6.2 DMA to external I/O

In the external I/O mode, DMA channel 1 supports 8-bit and 16-bit transfers. 16-bit transfers are only supported to/from 16-bit wide memories, and only with 16-bit alignment of data. The polarity of the external DMA request and DMA acknowledge signals are configurable.

The peripheral chip select signals $\overline{CS1}$ and $\overline{CS2}$ are used as data strobes for the external I/O.s:

Name:		Description:
Input:	EX_DREQ	External DMA request
Outputs:	EX_DACK	External DMA acknowledge
	$\overline{CS1}$	Write strobe for data to external I/O
	$\overline{CS2}$	Read strobe for data from external I/O

The number of waitstates during an external DMA transfer is controlled by the **io_ws** field of the **R_LATE_WS** register, see page 58. Note that external DMA transfers usually require extra waitstates compared to the normal memory cycles.

Transfers from an external I/O to the DRAM area will always result in at least one waitstate, to allow data from the I/O device to stabilize before $\overline{CAS}$ goes low.

1.6.3 Automatic program download ("flash-load")

DMA channel 0 is able to automatically load a program from the parallel input port to the system RAM after power up. This feature is enabled by keeping the input pin $\overline{\text{PR\_FLASH}}$ low during reset. In this case, 4096 bytes (1000 hex) are transferred to the system SRAM area, with start at address 40000002 hex, before the CPU starts to execute. This also forces the CPU to start execution at address 40000002 hex instead of starting at address 00000002.

1.7 RESET

1.7.1 Normal case

After reset, the CRIS CPU starts the execution at address 00000002. The following registers are initialized after reset:

Register	Value (hex)
VR	<Version number>
CCR	0000
DCR0	1000
IBR	00000000
DTP0	00000002

All other CPU registers have unknown values after reset.

1.7.2 Automatic program download

When the automatic program download ("flash-load") is enabled, the initial values of DCR0 and DTP0 change. After the program download has completed, the registers have the following values:

Register	Value (hex)
VR	<Version number>
CCR	0000
DCR0	0000
IBR	00000000
DTP0	40001002

Also, the CPU starts to execute at address 40000002 (hex) instead of 00000002.

1.8 VERSION IDENTIFICATION

Different versions of the CRIS CPU can be identified by reading the **Version Register** (**VR**). The version register is an 8-bit read-only register that contains the CPU version number. The following versions exist today::

Version:	Version number:	Marking:
ETRAX rev.1	00	GSC02PH364DH06 or GSX02PH364DH06
ETRAX rev.2	01	GSC02PH364DH08 or GSX02PH364DH08

2 PORTS

The ports include:

- One configurable Parallel output or Shared RAM port. (parallel port 1).
- One configurable Parallel input or Parallel output port. (parallel port 2).
- One asynchronous serial receiver/transmitter.
- General purpose port

The parallel input and output ports and the asynchronous serial port are able to use the DMA channels. It is also possible to feed data directly from the parallel input port to parallel output port 1.

2.1 PARALLEL (CENTRONICS) OUTPUT PORTS

Both parallel printer ports can be configured as output ports. In output mode, the ports can be configured to communicate with printers using various parallel printer protocols, including:

- IBM XT/AT compatible Centronics
- IBM PS/2 compatible Centronics
- Hewlett Packard Fast Mode
- IBM Fastbyte
- Bitronics, compatible with IEEE 1284 and HP Boise specifications.

2.1.1 Configuration

The operation mode is set by the **R_PAR_CONFIG** register for parallel port 1 and the **R_PAR2_CONFIG** register for parallel port 2, see page 81 and page 87. The parallel output ports have four basic operation modes:

- Centronics
- IBM Fastbyte
- Manual mode
- Reverse mode

In the Centronics and IBM Fastbyte modes, the strobe is generated automatically when the output data register is written. These modes can also be used for DMA transfers, see section 1.6, page 16. The Centronics mode can be further configured to await the printer ACK or to ignore ACK. The data setup time before strobe is configurable. For the Centronics mode, also the strobe length and the data hold time after the strobe can be configured. The internal registers for setting data setup time, strobe length and data hold time for the two ports are shown in the table below:

Parameter	Port 1 register	Port 2 register
Data setup time before strobe	R_PAR_DSETUP	R_PAR2_DSETUP
Strobe length	R_PAR_DSETUP	R_PAR2_DSETUP
Data hold time after strobe ***Note 1***	R_PAR_DHOLD	R_PAR2_DHOLD

Note 1: The port can also be configured to hold data until the printer ACK is received, by setting the **dob** bit in the **R_PAR_CTRL** or **R_PAR2_CTRL** register respectively.

The range for the register values is 0 - 31. The parallel output port timing depends on the periferal clock controlled by the clkx field in the **R_CLOCK_MODE** register, see page 61. The setup, strobe and hold times are given by:

```
(((reg. value + 1) * clkx divide factor) ±1) * system clock cycle time
```

In the manual mode, the strobe has to be generated by software. The manual mode gives the software full control of the port pins, allowing for implementation of printer protocols not compatible with standard Centronics or IBM Fastbyte.

The reverse mode is set by the **dir** bit in the **R_PAR_CONFIG** or **R_PAR2_CONFIG** register. This overrides the selection between Centronics, IBM Fastbytes or manual mode. The reverse mode is similar to the manual mode, but turns the port data outputs off regardless of the **dob** bit in the **R_PAR_CTRL**/**R_PAR2_CTRL** registers.

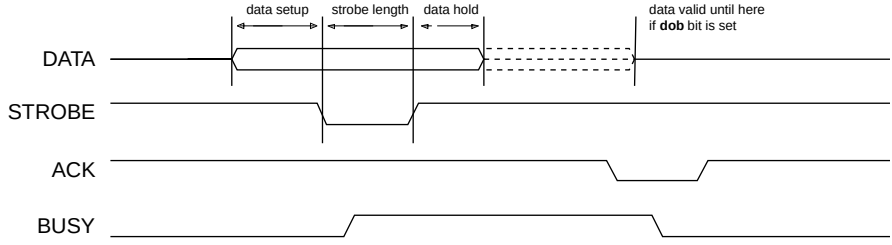
The manual and reverse modes allow for the implementation of Bitronics bidirectional protocol reverse transfers in both byte and nibble mode. The manual and reverse modes can not be used together with DMA.

2.1.2 Centronics and IBM Fastbyte operation

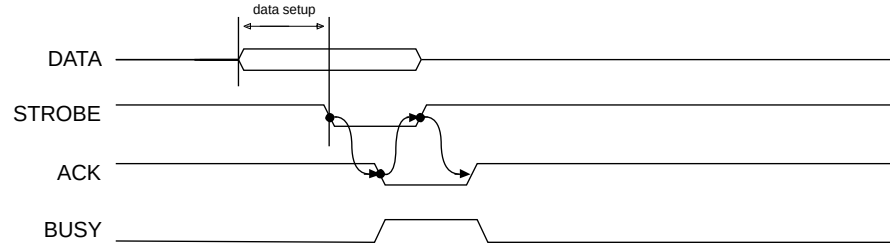
In the Centronics or IBM Fastbyte modes, a byte is transmitted to the printer by writing to the data register (**R_PAR_DATA** for port 1 and **R_PAR2_DATA** for port 2). The strobe is generated automatically. The AUTOFEED, INIT and SELECTIN outputs are controlled by the **R_PAR_CTRL** and **R_PAR2_CTRL** registers.

In the Centronics mode, the ports can be configured to await the printer ACK before the next transfer, or to ignore ACK. In the IBM Fastbytes mode, the strobe remains active until the printer sends an ACK. In both modes, the port requires SELECT active, and BUSY and PE inactive, before sending a strobe to the printer.

Centronics protocol



Fastbyte Protocol



The ports generate the Par1 or Par2 interrupt when they are ready to accept new data. The status of the ports can also be read on the **R_PAR_STAT** register for port 1 and the **R_PAR2_STAT** register for port 2.

2.1.3 Manual mode and reverse mode operation

In the manual mode, the software has full control of the port pins, including the strobe output. Data output is controlled by the **R_PAR_DATA** register for port 1 and **R_PAR2_DATA** register for port 2. The control outputs and the data output enable/disable is controlled by the **R_PAR_CTRL** and **R_PAR2_CTRL** registers. The control input signals can be read from the **R_PAR_STAT** and **R_PAR2_STAT** registers. The reverse mode is similar to the manual mode, but the output data is always turned off.

The byte mode reverse data in the Bitronics bidirectional protocol can be read from the **R_PAR_BACKCH** and **R_PAR2_BACKCH** registers. For parallel port 2, the direction of the port data is output on the PCH_DIR output.

2.1.4 Parallel output port pins

Each port has the following inputs and outputs. The pin names are given by adding a '1' or a '2' to the name, except for the data bits. :

Name:		Description:
Inputs:	$\overline{\text{ACK}}$	Acknowledge from printer
	BUSY	Busy signal from printer
	PE	Paper Empty signal from printer
	$\overline{\text{SELECT}}$	Select signal from printer
	$\overline{\text{FAULT}}$	Fault signal from printer
Outputs:	AUTO_FD	Autofeed signal to printer
	SEL_IN	Select in signal to printer
	STROBE	Strobe signal to printer
	INIT	Init signal to printer
I/O	PAR_D<7-0>	Data to/from printer, port 1
	PCH_D<7-0>	Data to/from printer, port 2

For compatibility with the Centronics and IBM Fastbytes protocols, the control outputs should be buffered with open collector buffers outside the ETRAX. For parallel port 2 the external buffers should be inverting. For parallel port 1, the polarity of the four control signals is controlled by the PAR_POL input. For PAR_POL low, the parallel port 1 control signals need inverting buffers.

For port 2, the direction of the port data is output on the PCH_DIR pin.

2.2 PARALLEL (CENTRONICS) INPUT PORT

The parallel printer port 2 can be configured as an input port, emulating the printer side of various parallel printer protocols, including:

- IBM XT/AT compatible Centronics
- IBM PS/2 compatible Centronics
- Hewlett Packard Fast Mode
- IBM Fastbyte
- Bitronics, compatible with IEEE 1284 and HP Boise specifications.

2.2.1 Configuration

The operation mode is set by the **R_PCH_CONFIG** register. The parallel input port has four basic operation modes:

- Centronics
- IBM Fastbyte
- Manual mode
- Reverse mode

In the Centronics and IBM Fastbyte modes, the BUSY and ACK are generated automatically. These modes can also be used for DMA transfers, see section 1.6, page 16. The ACK length (Centronics mode) or ACK minimum length (IBM Fastbytes mode) is controlled by the **R_PCH_ACKTIME** register. In Centronics mode, the ACK time is:

```
(reg. value + 1) * 8 * system clock cycle time
```

In IBM Fastbyte mode, the minimum ACK time is:

```
(reg. value + 1) * 4 * system clock cycle time
```

In the manual mode, the ACK has to be generated by software. The manual mode gives the software full control of the port pins, allowing for implementation of printer protocols not compatible with standard Centronics or IBM Fastbyte.

The reverse mode is set by the **dir** bit in the **R_PCH_CONFIG** register. This overrides the selection between Centronics, IBM Fastbytes or manual mode. The reverse mode is similar to the manual mode, but also allows for the implementation of Bitronics bidirectional protocol reverse transfers in both byte and nibble mode.

The reverse mode can not be used together with DMA. The manual mode may be used together with DMA, but no ACK is generated, and BUSY is not asserted between the transfers.

2.2.2 Centronics and Fastbyte operation

In Centronics and Fastbyte modes, BUSY is set and ACK is generated automatically when a byte is received. The presence of new data is indicated by the **dav** bit in the **R_PCH_STAT** register. The ACK length is set by the **R_PCH_ACKTIME** register, see section 2.2.1 above.

Input data is read from the **R_PCH_DATA** register. A read from this register removes BUSY and clears the **dav** bit. The **dav** bit of the **R_PCH_STAT** register is used internally as an interrupt request for the pch interrupt.

The FAULT, SELECT and PE outputs are controlled by the **R_PCH_CTRL** register. The ACK and BUSY bits of the register has no effect in the Centronics and Fastbytes mode. The status from the port can be read from the **R_PCH_STAT** register.

2.2.3 Manual mode operation

In the manual mode, the FAULT, SELECT, PE, ACK and BUSY outputs are controlled by the **R_PCH_CTRL** register. The presence of new data is indicated by the **dav** bit of the **R_PCH_STAT** register. The status from the port can be read from the **R_PCH_STAT** register.

Input data is read from the **R_PCH_DATA** register. A read to this register clears the **dav** bit. The **dav** bit of the **R_PCH_STAT** register is used internally as an interrupt request for the pch interrupt.

2.2.4 Reverse mode operation

In the reverse mode, the software has full control of the port pins. The direction of the port data is controlled by the **dob** bit of the **R_PCH_CTRL** register. Reverse data is controlled by the **R_PCH_BACKCH** register. The port data direction is output on the **PCH_DIR** pin.

A request for reverse mode operation is indicated by the **bid** bit of the **R_PCH_STAT** register. The **dav** bit of the **R_PCH_STAT** register is not set if the host tries to send data while the port is in reverse mode.

2.2.5 Parallel input port pins

The port has the following inputs and outputs:

Name:		Description:
Inputs:	$\overline{\text{PCH\_AUTO\_FD}}$	Autofeed signal from host
	$\overline{\text{PCH\_SEL\_IN}}$	Select in signal from host
	$\overline{\text{PCH\_STROBE}}$	Strobe signal from host
	$\overline{\text{PCH\_INIT}}$	Init signal from host
Outputs:	PCH_ACK	Acknowledge signal to host
	$\overline{\text{PCH\_BUSY}}$	Busy signal to host
	$\overline{\text{PCH\_PE}}$	Paper Empty signal to host
	$\overline{\text{PCH\_SELECT}}$	Select signal to host
	$\overline{\text{PCH\_FAULT}}$	Fault signal to host
	$\overline{\text{PCH\_DIR}}$	Data direction signal
I/O	PCH_D<7 - 0>	Data from/to host

The control outputs should be buffered with inverting open collector buffers outside of ETRAX.

2.3 SHARED RAM INTERFACE

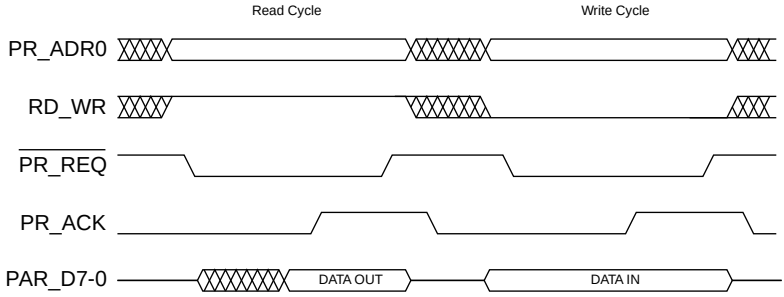
The shared RAM interface handles byte-wide transfers between an external device and the system SRAM. The shared RAM interface operates with standard SRAM. The data transfers are controlled by an asynchronous handshake protocol. One interrupt signal in each direction is also provided.

The address from the external device is supplied to the SRAM through external multiplexers, except for address bit 0, which is multiplexed internally. The shared RAM area is located within the lowest 1 Mbyte of the SRAM area used by the CPU, and the SRAM bus width may be either 8 or 16 bits. During a shared RAM access, address bits A19 - A12 are controlled by the **R_MIO_ADDR** register (see page 72), while the other address bits out from the ETRAX (except A0) are output as 0's.

The Shared RAM interface is enabled/disabled and the interrupt signals are controlled by the **R_MIO_MODE** register, see page 72 .

Name:		Description:
Inputs:	$\overline{\text{PR\_REQ}}$	Transfer request, active low
	RD_WR	Read/write select. Read = 1, write = 0.
	$\overline{\text{INTIO}}$	Interrupt from the external device.
	PR_ADR0	Address bit 0 from the external device.
Outputs:	PR_ACK	Transfer acknowledge.
	PR_INT	Interrupt to the external device.
	A_SEL	Select signal for the address mux.
I/O	PAR_D<7 - 0>	Data to/from the external device.

The polarity of the PR_ACK and PR_INT signals is controlled by the input PAR_POL. For PAR_POL high, the PR_ACK and PR_INT signals are active high.



2-1 Shared RAM handshake protocol

The interrupt signal $\overline{\text{INTIO}}$ from the external device is sampled by the peripheral clock controlled by the clkx field of the **R_CLOCK_MODE** register, see page 61 . $\overline{\text{INTIO}}$ requires a negative pulse with at least the length:

```
(clkx divide factor + 1) * system clock cycle time
```

The interrupt signal PR_INT to the external device is generated as a pulse with a length of:

```
((clkx divide factor * 4) ± 1) * system clock cycle time
```

2.4 ASYNCRONOUS SERIAL PORT

This block contains one complete asynchronous serial receiver/transmitter with full buffering and parity control, and with one handshake signal in each direction. The receiver/transmitter supports baud rates from 300 up to 115200 baud.

2.4.1 Connection to input/output pins

The inputs and outputs of the serial port are multiplexed to two different sets of input/output pins, making it possible to alternate between two different external channels. When not used for the serial port the TXD and RTS outputs can be used as general purpose outputs. TXD2 can also be configured as receiver or transmitter baudrate clock output, or system timer output, see section 3.2.3, page 31. The RXD and CTS inputs can be read by software. The connection of the serial port to the different input/output pins is controlled by the **R_SER_CONNECT** register. The input pins can be read from the **R_P2_DATA** register, see page 74.

Name:		**Description:**
Inputs:	RXD1	Receive data, channel 1.
	$\overline{\text{CTS1}}$	Clear to send, channel 1.
	RXD2	Clear to send, channel 1.
	$\overline{\text{CTS2}}$	Clear to send, channel 2.
Outputs:	TXD1	Transmit data, channel 1.
	$\overline{\text{RTS1}}$	Request to send, channel 1.
	TXD2	Transmit data, channel 2.
I/O	$\overline{\text{RTS2}}$	Request to send, channel 2.

2.4.2 Baud rate selection

The baud rates are set by the **R_SER1_BAUD** register. The baud rates can be set independently for the receiver and transmitter. The available baud rate selections are shown in the table below.

Baud rates:	
115200	4800
57600	2400
38400	1200
19200	600
9600	300

2.4.3 Operation modes

The serial port receiver and transmitter can be configured for odd, even or no parity, 7 or 8 data bits, and 1 or 2 stop bits. The transmitter can be set to handle $\overline{\text{CTS}}$ automatically, or to ignore $\overline{\text{CTS}}$. The receiver $\overline{\text{RTS}}$ output is controlled by software.

The modes for the receiver and transmitter can be set independent of each other. The **R_SER1_IN_CFG** register sets the mode for the receiver, and the

R_SER1_OUT_CFG register sets the mode for the transmitter. The status of the serial receiver and transmitter can be read from the **R_SER1_STAT** register, see page 89.

2.4.4 Transmit operation

Transmission is started by writing a data byte in the **R_SER1_DOUT** register. The serial transmitter generates a serial out interrupt, see page 88, when the transmitter is ready to accept new data. The transmitter status can also be polled on the **R_SER1_STAT** register.

When the automatic $\overline{\text{CTS}}$ handling is enabled, transmission is stopped while $\overline{\text{CTS}}$ is high. If $\overline{\text{CTS}}$ goes high during transmission, the transmission is stopped after the ongoing byte.

The serial transmitter can also use DMA transfers, see section 1.6, page 16.

2.4.5 Receive operation

When the receiver has received a data byte, it generates a serial in interrupt and sets the **dav** bit in the **R_SER1_STAT** register, see page 89. The input data is available in the **R_SER1_DIN** register. Reading the data register clears the interrupt and the **dav** bit in the **R_SER1_STAT** register.

RTS is controlled by the **rts** bit in the **R_SER1_IN_CFG** register.

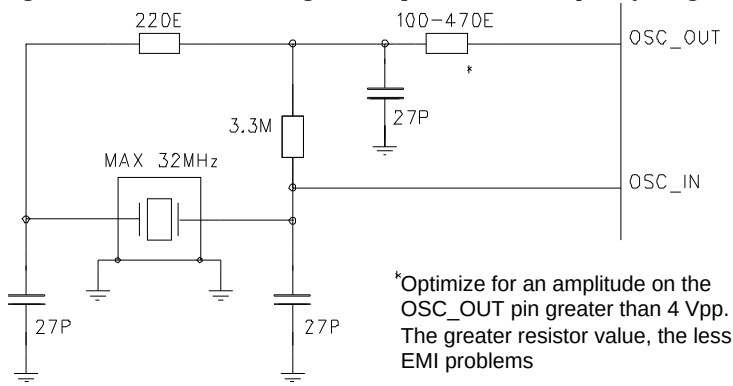
2.5 GENERAL PURPOSE PORT

The general purpose port P1<7:0> contains 8 software controlled I/O pins. The direction of each pin is individually programmable. The direction is controlled by the **R_P1_DIR** register, with 1's for the outputs and 0's for the inputs. The output pins are controlled by the **R_P1_DATA_OUT** register. The inputs are read from the **R_P1_DATA_IN** register. Port pins directed as outputs can be read back through the **R_P1_DATA_IN** register.

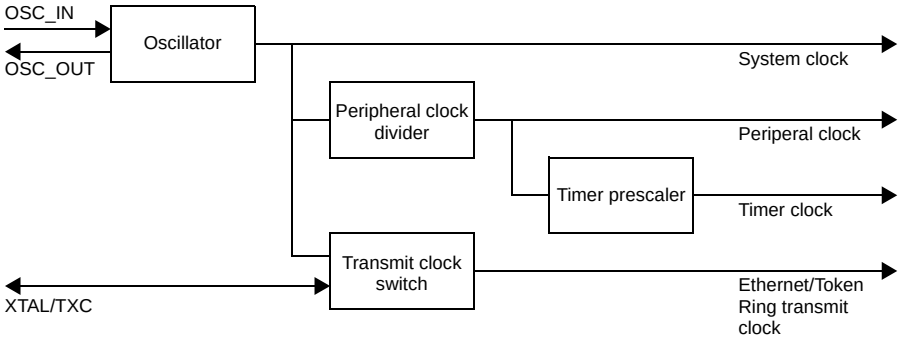
3 TIMING

3.1 CLOCK GENERATOR

This block contains the active parts of a crystal controlled oscillator for the basic clock signal. The oscillator is designed to operate in the frequency range 18 - 32 MHz.



The clock generator block also supplies the different internal clock signals derived from the basic clock frequency. The clock generator is controlled by the **R_CLOCK_MODE** register, see page 61. A block diagram of the clock generator is shown below.



3.1.1 Transmit clock switch

The Transmit clock switch selects between internal or external transmit clock for the Ethernet/Token Ring block, and also selects divide by 1 or divide by 4 operation. Ethernet requires external clock and divide by 1.

Token Ring normally uses the internal clock, with divide by 1 for 16 Mb/s and divide by 4 for 4 Mb/s. In this case the system clock frequency must be 32 MHz. Selecting internal transmit clock also turns the XTAL/TXC pin to an output. The transmit clock is output on the XTAL/TXC pin as long as the **FRAQ** bit in the **R_ANALOG** register (see page 99) is set.

3.1.2 Peripheral clock divider

The peripheral clock divider generates a divided clock used by the parallel output ports and the shared RAM interrupt logic. The peripheral clock is also used as an input to the timer prescaler. The divide factor is programmable from 3 up to 6.

3.1.3 Timer Prescaler

The timer prescaler further divides the peripheral clock, to generate an input clock to the timer block. The timer block needs a 921.6 kHz clock to be able to generate the baudrate clocks specified for the serial port. The timer prescaler divide factors are designed to give a 921.6 kHz timer clock together with commonly used system clock frequencies.

Prescale mode	Prescale divide factor
0	7.22
1	6.50
2	6.78
3	6.94
4	7.00
7	Stopped

The table below shows how to set up the peripheral clock and prescale divide factors to get the desired 921.6 kHz timer clock for different system clock frequencies:

System Clock Frequency	Peripheral clock divide factor	Prescale mode
18.0 MHz	3	1
18.7 MHz	3	2
19.2 MHz	3	3
19.4 MHz	3	4
20.0 MHz	3	0
24.0 MHz	4	1
25.0 MHz	4	2
25.6 MHz	4	3
25.8 MHz	4	4
26.6 MHz	4	0
30.0 MHz	5	1
31.2 MHz	5	2
32.0 MHz	5	3

3.2 TIMERS

This block contains one baud rate timer and one programmable system timer. The input clock for the timers is generated by the clock generator block. The timer block is controlled by the registers **R_TIMER_DATA**, **R_TIMER_MODE** and **R_TIMER_INTA**, see page 92 to page 94.

3.2.1 Baud rate timer

This timer generates fixed baud rate clocks from which the serial port receiver and transmitter can select its clock signal. The selectable baud rates are listed in section 2.4.2, page 27. The baud rates are selected by the **R_SER1_BAUD** register, see page 89.

The baudrate timer also generates the clocks for the system timer, the Token Ring TVX and TRR timers, for the Token Ring $\overline{\text{NSRT}}$ output and for the DRAM refresh. This means that the baud rate timer has to be enabled when these functions are to be used.

3.2.2 Programmable system timer

This is a programmable system timer used to generate the timer tick interrupt. This timer can be programmed to an 8-bit count value, and can use any of the available serial baud rate clocks as the basic clock. It is also possible to run the system timer on the input clock to the timer block (921.6 kHz). This gives a programmable range from < 10 µs to 850 ms.

At terminal count, the timer sets the timer tick interrupt, and restarts. The timer tick interrupt is reset by writing to the **R_TIMER_INTA** register. The data written is don't care.

The system timer divide factor range is from 2 to 256. A divide factor of 256 is acheived by setting the count value to 0.

Example: To get a 10 ms timer tick, set the timer basic frequency to 300 Hz (**freq_select** field of the **R_TIMER_MODE** register set to 1), and the count value to 3 (**R_TIMER_DATA** register set to 3).

3.2.3 Timer output pin

The TXD2 output can be reconfigured as an output for the selected clock for the serial receiver or transmitter, or for the system timer. The TXD2 operation is controlled by the **R_P2_CONFIG** register, see page 87. Selection of any of the possible timer outputs overrides the TXD2 configuration in the **R_SER_CONNECT** register.

The serial transmit clock corresponds to the selected serial transmit baudrate. The serial receive clock is 8 times the selected serial receive baudrate. The serial receive or transmit clock is output as a symmetrical square wave.

When the system timer is selected, a positive pulse is output each time the timer reaches its terminal count. The pulse length is one clock cycle of the 921.6 kHz input clock for the timer block.

3.2.4 Timer initialization

In order to make the timer block operate properly, the initialization must be performed in the following sequence:

1. Start the timer prescaler, see section 3.2 above. Load the **R_TIMER_DATA** register with the desired value. Load the **R_SER1_BAUD** register with valid baudrate selections.
2. Wait at least 45 system clock cycles.
3. Start the baudrate timer and the system timer by loading the **R_TIMER_MODE** register. (The baudrate timer may be started without starting the system timer.)

Once initialized, the registers may be changed in any order. A new system timer count value (**R_TIMER_DATA**) takes effect when the timer reaches its terminal count.

4 TOKEN RING AND ETHERNET INTERFACE

This block is a network controller, configurable for Token Ring or Ethernet operation. For Token Ring, ETRAX is compatible with IEEE 802.5 and IBM specifications, and interfaces directly to the Texas Instruments TMS38054 Ring Interface Circuit. For Ethernet, ETRAX is compatible with IEEE 802.3 and Ethernet II specifications, and interfaces directly to the National Semiconductor DP8391A Serial Network Interface, the Intel 82503 Dual Serial Transceiver and compatible circuits.

Token Ring or Ethernet operation is selected by the ethernet bit in the **R_TR_MODE1** register. This bit should only be changed when the network controller is in the reset state (**etr_reset** bit in the **R_TR_MODE2** register set to 0).

Receive and transmit data is stored in ring buffers in system memory. The ring buffer control logic is common to both Token Ring and Ethernet.

4.1 TOKEN RING OPERATION

The token-ring protocol handler implement time critical parts of the IEEE 802.5 MAC protocol. This includes Frame Transmission, Token Transmission, Stripping, Frame Reception, Priority Operation, TRR timer and TVX timer, burst error detection and elimination, Frame Check Sequence generation and checking, detection and elimination of circulating frames and priority tokens. The token-ring protocol handler also provide the latency- and elastic buffers. In IEEE 802.5 terms this means the Receive Actions, the Operational Finite-State Machine and part of the Active Monitor Finite-State Machine.

The parts of the IEEE 802.5 MAC protocol that is based on receiving and transmitting MAC frames, status counters and slow timers is handled by interrupt driven software. Under normal conditions the MAC protocol generate one AMP or SMP frame per station every seven seconds.

4.1.1 Transmit operation

When the transmission of a frame is requested by the software, the token-ring protocol handler waits for a free token with low enough priority. The transmission starts by modifying the Token bit in the Access Control field (AC) to change the token to a frame, and then data is sent from the ring buffer. This user supplied data must include, in order, the Access Control field, which is read in advance by the ring buffer handler so the protocol handler knows the frame priority, Frame Control field (FC), Destination (DA) and Source address (SA), and frame Data. After all data is sent the transmitter appends the Frame Check Sequence (FCS), the Ending Delimiter (ED) and finally the Frame Status field (FS).

If the ring buffer handler has more frames to transmit, the protocol handler will append them immediately after the first one, as long as the command byte of the next frame is

0x10, the **new_token** bit of the **R_TR_CMD** register isn't set and the THT timer hasn't expired.

When there are no more frames to transmit, or the command byte of the next frame is 0x14, or the **new_token** bit of the **R_TR_CMD** register is set, or the THT timer has expired, the protocol handler will start to send fill until all transmitted frames has been received and then release a new token. If the **etr** bit of the **R_TR_MODE1** register is set, a new token will be released immediately, without waiting for the transmitted frames to return.

When a token has been released, but the transmit ring buffer isn't empty, the transmission will be resumed without software intervention, as soon as the transmitter again finds a token with low enough priority.

4.1.2 Receive operation

The receive operation is started when the receiver detects a Starting Delimiter (SD) followed by an Access Control (AC) with the Token bit set to 1. The receiver strips off the Starting Delimiter, and transfers the data to the ring buffer handler. The receiver is clocked by the RCLK/RXC input. The receiver compares the Destination Address field (byte 2 to 7) with the addresses stored in the station address and functional/group address registers. If the address does not match, the receive operation is aborted, and the ring buffer handler discards the packet.

During reception, a local CRC is generated and compared with the FCS field of the incoming packet. Packets with CRC errors are rejected. When the Frame Status bytes passes, changes are made to indicate if the destination address was recognized and if the frame was copied (A and C bits in FS). If an error is detected, the E bit in the Ending Delimiter is set.

4.1.3 Station address

The individual address of the station is set by the **R_MA0** to **R_MA5** registers. The address field of a frame is 48 bits, where bit 47 indicates individual (bit 47 = 0) or functional/group (bit 47 = 1) address. A frame with an individual address is accepted if the following expression is true:

```
addr<47>     == 0 &&
addr<46:40>  == R_MA0<6:0> &&
addr<39:32>  == R_MA1<7:0> &&
addr<31:24>  == R_MA2<7:0> &&
addr<23:16>  == R_MA3<7:0> &&
addr<15:8>   == R_MA4<7:0> &&
addr<7:0>    == R_MA5<7:0>
```

Note: Bit 47 in a station address always is zero and therefore bit 7 of **R_MA0** must also be zero.

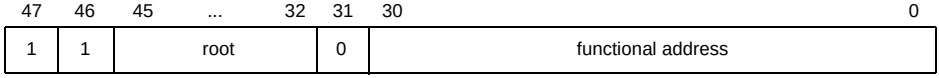
The address recognition can be disabled by setting the **enable_adr** bit in the **R_TR_MODE2** register to 0. In this case only broadcast frames are accepted.

The station address should be changed only when the **etr_reset** bit (bit 2) of the **R_TR_MODE2** register is low or when the **enable_adr** bit is low.

If the **promiscuous** bit in the **R_TR_MODE1** register is set, the address comparison logic will accept frames to all addresses. The **promiscuous** bit should only be changed when the **etr_reset** bit of the **R_TR_MODE2** register is low.

4.1.4 Functional/group address

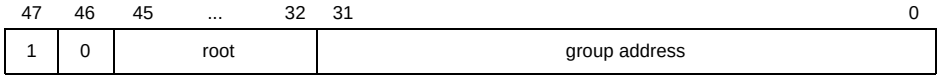
The group addresses are divided into three types: functional, universal and local group addresses. The 'all_roots' mode bit, **R_GA4**<0>, selects for all group addresses whether all values of the root field should be accepted or if only 0b00000000000000 should be accepted.:



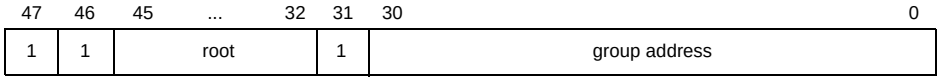
4-1 Functional address

The functional address is bit significant (i.e. each bit in the address field corresponds to one functional address). A frame is copied if the following condition is met.

```
addr<47:46>    == 0b11 &&
( addr<45:32> == 0b00000000000000 || all_roots == 1 ) &&
addr<31>       == 0 &&
(
( R_GA4<7:1> & addr<24:30> ) ||
( R_GA5<7:0> & addr<16:23> ) ||
( R_GA6<7:0> & addr<8:15>  ) ||
( R_GA7<7:0> & addr<0:7>   )
)
```



4-2 Universal/local group addresses:



4-3 Local group address:

The group address can be interpreted either as a bit significant address, like the functional address, or individual as the station address. This is determined by a **mode** bit in the **R_GA0** register.

R_GA0<0> - individual	
1	Group address is interpreted as an individual address.
0	Group address is interpreted as a bit significant address.

If group addressing is in bit significant mode a frame is copied if the following expressions are true.

Local bit significant group address:

```
addr<47:46>   == 0b11 &&
( addr<45:32> == 0b00000000000000 || all_roots == 1 ) &&
addr<31>      == 1 &&
(
( R_GA0<7:1> & addr<24:30> ) ||
( R_GA1<7:0> & addr<16:23> ) ||
( R_GA2<7:0> & addr<8:15>  ) ||
( R_GA3<7:0> & addr<0:7>   )
)
```

Universal bit significant group address:

```
addr<47:46>   == 0b10 &&
( addr<45:32> == 0b00000000000000 || all_roots == 1 ) &&
(
( R_GA0<7:0> & addr<24:31> ) ||
( R_GA1<7:0> & addr<16:23> ) ||
( R_GA2<7:0> & addr<8:15>  ) ||
( R_GA3<7:0> & addr<0:7>   )
)
```

Note Since **R_GA0**<0> also is mode bit 'individ' which is 0 in this case, the universal bit significant group address with addr<31> can't be received.

If group addressing is in individual mode a frame is copied if the following expressions are true.

Local individual group address:

```
addr<47:46>   == 0b11 &&
( addr<45:32> == 0b00000000000000 || all_roots == 1 ) &&
addr<31>      == 1 &&
R_GA0<7:1>    == addr<30:24> &&
R_GA0<7:0>    == addr<23:16> &&
R_GA0<7:0>    == addr<15:8>  &&
R_GA0<7:0>    == addr<7:0>
```

Universal individual group address:

```
addr<47:46>   == 0b10 &&
( addr<45:32> == 0b00000000000000 || all_roots == 1 ) &&
R_GA0<7:1>    == addr<30:24> &&
R_GA0<7:0>    == addr<23:16> &&
R_GA0<7:0>    == addr<15:8> &&
R_GA0<7:0>    == addr<7:0>
```

Note In this case the addr<31> bit isn't used in the comparison.

Group address registers must only change value when **$\overline{\text{etr\_reset}}$** (**R_TR_MODE2** bit 2) is low or when **enable_adr** (**R_TR_MODE2**<4>) is low.

4.1.5 Elastic buffer

The Active monitor requires an elastic buffer in order to compensate for phase gitter and provide the minimum ring latency. The elastic buffer has a total length of 56 bits, divided in a fixed portion of 24 bits and an elastic portion of 32 bits.

The elastic buffer is inserted by setting the **elastic_buffer** bit in the **R_TR_MODE2** register. When inserted, the buffer length is initialized to the middle of the elastic portion of the buffer. If the buffer over- or underflows (buffer length goes above 56 bits or below 24 bits), the buffer reinitializes, and a frequency error interrupt is generated.

When the elastic buffer is enabled, the transmitter uses the transmit clock from the Clock Generator (see section 3.1, page 29) instead of the recovered network clock from the RCLK/RXC input.

4.1.6 MAC protocol support

The Medium Access Control (MAC) protocol controls which station on the LAN that gains access to the LAN media and how to use the media. In token-ring mode the ETRAX implement time critical parts of the IEEE 802.5 standard, non time critical parts of the standard are implemented in software.

4.1.7 Interrupts

Network interrupt status The Network controller block communicates with the MAC support software through eight interrupts. These interrupts are OR:ed together to generate bit 1 of the internal interrupt vector number, see section 6.1, page 52.

The individual interrupts are separated by the **R_ETR_STATUS** register. The function of the **R_ETR_STATUS** register is controlled by the **rd_intr_status** bit of the **R_TR_MODE2** register. When the **rd_intr_status** bit is low, the **R_ETR_STATUS** register shows the interrupt bits transparently. On a low-to-high transition of **rd_intr_status**, the current interrupt status is latched into **R_ETR_STATUS**, and the interrupts are cleared. If new interrupts occur while **rd_intr_status** is high, they will not appear in the **R_ETR_STATUS** register until the **rd_intr_status** bit is set low again.

The **rd_intr_status** requires an internal synchronization time of 16 network bit times before it takes effect.

Frequency error interrupt The frequency error interrupt is generated when the Elastic buffer over- or underflows. This interrupt is used to implement the Frequency Error Counter. In Standby Monitor mode, the intensity of the frequency error interrupt is a measure of the frequency difference between the internal bit clock and the recovered receive clock.

Lost frame interrupt The 'lost_frame' interrupt is generated when the protocol handler is waiting for transmitted frames to return and the TRR-timer expires. This interrupt is used to implement the Lost Frame Error Counter (IEEE-802.5 section 3.8.3 and transitions T22 and T32). Note that the 'lost_frame' interrupt isn't valid when the MAC support software is in state A1.

Token error interrupt The 'token_error' interrupt is generated when the protocol handler is in Active Monitor mode (and only then) and finds a token or frame with the Monitor (M) bit set to one. This interrupt is used to signal the MAC support software that the protocol handler has executed transition A01 in hardware. Note that this isn't the only cause to increment the Token Error Counter, it only indicates transition A01.

Abort transmitted interrupt The 'abort_trans' interrupt is generated when the protocol handler has started transmission of a frame and finds out that the token used for that transmission did not end with a correct Ending Delimiter (IEEE-802.5 transition T11). This interrupt is used to implement the Abort Delimiter Transmitted Counter (IEEE-802.5 section 3.8.5).

Line error interrupt The 'line_error' interrupt is generated when the protocol handler finds a frame or token with error and the the Error (E) bit in that frame or token isn't set. At the same time the E bit in the frame or token is set. This interrupt is used to implement the Line Error Counter (IEEE-802.5 section 3.8.1 and 4.2.1)

TVX expired interrupt The 'tvx_exp' interrupt is generated when the TVX-timer expires. See description of **R_TVX** register for TVX-timer operation. (page 97)

TRR expired interrupt The 'trr_exp' interrupt is generated when the TRR-timer expires. See description of **R_TRR** register for TRR-timer operation. (page 98)

Burst error interrupt The 'burst_error' interrupt is generated when the protocol handler detects loss of transitions in receive data for four consecutive half bits. At the same time a transition is inserted, converting burst-five to burst-four errors. This interrupt is used to implement the Burst Error Counter (IEEE-802.5 section 3.8.3).

4.1.8 Physical layer interface

ETRAX interfaces directly to the Texas Instruments TMS38054 Ring interface circuit. ETRAX has dedicated pins for the data and clock signals, the insert, speed switch, frequency acquisition and ready functions.

Wire fault and wrap signals do not have dedicated pins. These signals have to be connected through the general purpose port or through external logic.

Insert The ring insert function is controlled by the **insert** bit of the **R_TR_MODE2** register. When the insert bit is set, a 2400 Hz square wave is output on the $\overline{\text{NSRT}}$ output. This results in insertion into the ring. When the insert bit is cleared, the $\overline{\text{NSRT}}$ output is constantly high. The 2400 Hz insert frequency is generated by the baudrate timer (see section 3.2.1, page 31), and therefore the baudrate timer has to be started before the insert bit is set.

Speed switch ETRAX provides a speed switch output, SPSW, to set the ring speed to the Ring interface circuit. The speed switch output is controlled by the spsw bit in the **R_ANALOG** register. Note that this bit does not control the internal clock divide.

Frequency acquisition ETRAX provides a frequency Acquisition output, FRAQ, to set the ring interface circuit to phase or frequency acquisition mode. The frequency acquisition output is controlled by the fraq bit in the **R_ANALOG** register.

Ready ETRAX provides a ready input, $\overline{\text{REDY}}$, to read in the ready status from the Ring interface circuit. The ready status is available on the **sync** bit of the **R_REC_STATUS** register.

4.2 ETHERNET OPERATION

4.2.1 Transmit operation

The transmitter reads parallel data from the ring buffer, and serializes it for transmission. The transmitter is clocked by the transmit clock (XTAL/TXC) input. At the beginning of each packet, the transmitter appends the 62-bit preamble sequence and the 2-bit Start of Frame Delimiter (SFD). At the end of the packet, the transmitter appends the 32-bit Frame Check Sequence (FCS). In the case of a collision, the transmitter sends out a 32-bit JAM pattern of all 1's.

Note It is the responsibility of the software to ensure that the length of the packet is within the limits of the IEEE 802.3 specification. ETRAX does not append pad bytes for short packets or check for oversized packets.

4.2.2 Collision recovery

When the network controller detects a collision, the **retry_count** field of the **R_TR_STATUS** register is incremented. If the **retransmit** bit in the **R_TR_CMD** register is set, the transmitter performs up to 15 transmission retries, with random exponential backoff according to the IEEE 802.3 protocol. After 15 unsuccessful retries (or directly after the first collision, if the retransmit bit is cleared), the transmitter stops, and issues an Excessive retry interrupt, see section 4.3.10, page 45.

After a successful transmission of a packet, the **retry_count** field of the **R_TR_STATUS** retains its value until the transmission of the next packet is initiated.

4.2.3 Receive operation

The receive operation is activated when the Carrier Sense (CRS) input is asserted. The receiver deserializes the incoming data , strips off the preambler and Start of Frame Delimiter (SFD), and transfers the data to the ring buffer handler. The receiver is clocked by the RCLK/RXC input. The receiver compares the Destination Address Field (the 6 first bytes of the packet) to the addresses stored in the station address and group address registers. If the address does not match, the receive operation is aborted, and the ring buffer handler discards the packet.

During reception, a local CRC is generated and compared with the FCS field of the incoming packet. Packets with CRC errors are rejected. The receiver checks minimum frame size and ignores short frames as collision fragments. ETRAX does not check for oversized packets.

ETRAX can be configured to disable the receive operation while transmitting, or to allow simultaneous receive and transmit operation. This is set by the **loop_back** bit in the **R_TR_MODE1** register, where a '0' allows simultanuos receive and transmit. The **loop_back** bit should only be changed while the network interface is in the reset state.

4.2.4 Station address

The individual address of the station is set by the **R_MA0** to **R_MA5** registers. The address field of a frame is 48 bits, where bit 47 indicates individual (bit 47 = 0) or group (bit 47 = 1) address. A frame with an individual address is accepted if the following expression is true:

```
addr<47>    == 0 &&
addr<47:40> == R_MA0<7:0> &&
addr<39:32> == R_MA1<7:0> &&
addr<31:24> == R_MA2<7:0> &&
addr<23:16> == R_MA3<7:0> &&
addr<15:8>  == R_MA4<7:0> &&
addr<7:0>   == R_MA5<7:0>
```

Note The normal way of displaying ethernet addresses is:

```
addr<40:47>:addr<32:39>:addr<24:31>:addr<16:23>:addr<8:15>:addr<0:7>
```

Bit 47 in a station address always is zero and therefore bit 7 of **R_MA0** must also be zero.

The station address should be changed only when the **etr_reset** bit (bit 2) of the **R_TR_MODE2** register is low.

If the promiscuos bit in the **R_TR_MODE1** register is set, the address comparison logic will accept frames to all addresses. The promiscous bit should only be changed when the **etr_reset** bit of the **R_TR_MODE2** register is low.

4.2.5 Group address

An ethernet group address has bit 47 of the address field set to 1. In ethernet mode the five least significant bits of the internal crc at the time the destination address has been received is used to point to a bit in one of the group address registers **R_GA0** to **R_GA7**. If this bit is set, the packet is accepted.

This mechanism is used as an address filter to reduce the amount of packets that has to be handled in software. By setting all bits in all group address registers to 1, the address filtering is disabled and all group addresses are received.

A frame is accepted if the following expression is true:

```
GA<crc(addr<47:0>)<0:5>> == 1
```

Where

```
GA<63:0> = R_GA7<7:0>.R_GA6<7:0>.R_GA5<7:0>.R_GA4<7:0>.
R_GA3<7:0>.R_GA2<7:0>.R_GA1<7:0>.R_GA0<7:0>
```

The group address should be changed only when the **$\overline{\text{etr\_reset}}$** bit (bit 2) of the **R_TR_MODE2** register is low.

4.3 RECEIVE AND TRANSMIT RING BUFFERS

4.3.1 Receive ring buffer size and placement

The receive ring buffer is placed within the first 2 Mbyte of the system SRAM area starting at 40000000 hex. Address bit 20 - 13 of the ring buffer start address are set by the **R_REC_POS** register. The ring buffer can have one of four different sizes, set by the **rec_buf_size** field of the **R_RT_SIZE** register:

rec_buf_size:	Receive ring buffer size:
00	8 kbyte
01	16 kbyte
10	32 kbyte
11	64 kbyte

The ring buffer can start at an address that is a multiple of its size. (An 8 k buffer can start at 8 k bundaries, a 16 k buffer at 16 k boundaries etc.) The **R_REC_POS** register should be set up to match this alignment requirement.

4.3.2 Receive packet format

The packets are stored in the ring buffer in the following format:

Lower address — higher address

STATUS 1 byte	not used 1 byte	ENDPTR 2 bytes	PACKET_DATA n bytes

The **STATUS** field shows the status of the received packet:

Status(hex)	Description
00	No packet available. Previous packet was the latest completely received one.
01	A complete packet is available, and there is room for more packets in the ring buffer.
02	(reserved, not used)
03	A complete packet is available, and this packet caused a buffer full condition.
04-FF	(reserved, not used)

The **ENDPTR** field shows bit 15 - 0 of the buffer address following the last byte of the packet.

The **PACKET_DATA** contains all received bytes of the packet, including the FCS field for Ethernet, and the AC, FC, FCS, ED and FS fields for Token Ring. The J and K symbols in the ending delimiter (ED) field are interpreted as 1 and 0 respectively.

Each packet starts at a 256-byte boundary. After the end of the packet, the rest of the current 256-byte block is skipped. As a special case, if the packet ends at a 256-byte boundary, the following 256-byte block is skipped.

4.3.3 Receive ring buffer operation

Start packet receive When the receiver detects an incoming packet, it starts to write the received bytes to the ring buffer. The first data byte is placed at position 04 in the 256-byte block following the end of the previous packet.

Packet abort If, during the reception, the receiver hardware recognizes the packet as not being adressed to the unit, or an error is detected, the current packet is aborted, and the internal pointers are restored to their previous values.

Packet accept and receive If the received packet is destined to the unit, and is error free, the reception continues until the end of the packet. The incoming bytes are stored sequentially. If the packet overflows the physical end of the ring buffer, it wraps around and continues.

Congestion check For each byte received, the ring buffer logic checks bit 15 - 8 of the current ring buffer position against the **R_REC_END** register. If a match is found, the packet is aborted and the **buffer_full** bit of the **R_REC_STATUS** register is set. The buffer full condition will also generate a Buffer Full interrupt. The Buffer Full interrupt is cleared by writing a "1" to the **clr_buf_full** bit of the **R_REC_MODE** register.

However, the packet is continued if a match occurs after the receiver has detected the End of Packet internally. This means that there has to be at least one free 256-byte block beyond the point given by **R_REC_END**, to accomodate for bytes pipelined in the receiver. The buffer full status bit is set anyway.

Also, in the Token Ring case, if a MAC frame is detected that requires express buffering, the packet will be stored regardless of the buffer full status bit. This might result in that the MAC frame overwrites previously received packets.

Packet complete After the complete packet is received, the **STATUS** and **ENDPTR** fields of the received packet are stored at the beginning of the packet. Then a 0 is written in the first byte of the 256-byte block following the end of the packet. These 4 bytes will be written in one sequence, without intermediate CPU cycles, to avoid the possibility that the software reads values that are only partly updated. Finally, the receiver issues a Packet Received interrupt to the CPU. The Packet Received interrupt is cleared by writing a "1" to the **clr_pkt_recvd** bit of the **R_REC_MODE** register.

Buffer release When the software has processed a packet, it releases the buffer area by setting the **R_REC_END** register to match address bit 15 - 8 of the last complete 256 byte block of the packet. This means that the software has to release the packets in the same order as they arrive.

4.3.4 Initialization.

At initialization, the software must write 0 at position 0 of the ring buffer, and set the **R_REC_END** register to correspond to the second last 256 byte block of the buffer. Also the **R_REC_POS** and **R_RT_SIZE** registers must be initialized. The **reset_rec** bit (bit 0) of the **R_REC_MODE** register is set to 1 after system reset. To enable the receive buffer, write 6 to the **R_REC_MODE** register. This will start the receive ring buffer and also clear any pending Buffer Full or Packet Received interrupts.

4.3.5 Transmit ring buffer size and placement

The transmit ring buffer is placed within the first 2 Mbyte of the system SRAM area starting at 40000000 hex. Address bit 20 - 13 of the ring buffer start address are set by the **R_TR_POS** register. The ring buffer can have one of four different sizes, set by the **tr_buf_size** field of the **R_RT_SIZE** register:

tr_buf_size:	Transmit ring buffer size:
00	2 kbyte
01	4 kbyte
10	8 kbyte
11	64 kbyte

The ring buffer can start at an address that is a multiple of 8 kbyte, except for 64 k ring buffers that only can start at 64 k boundaries. For 64 k buffer size, the three lower bits of the **R_TR_POS** register should be set to 0.

4.3.6 Transmit packet format

The packets are stored in the ring buffer in the following format:

Lower address higher address

COMMAND 1 byte	not used 1 byte	ENDPTR 2 bytes	PACKET_DATA n bytes

The **COMMAND** byte contains a command that is interpreted by the transmitter when the transmission of the packet is started.

Command(hex)	Description
00	No packet available for transmission.
10	Packet available for transmission. In Token Ring operation, the packet could be sent immediately after the previous one, without awaiting a fresh token. Note that if the new_token bit in the R_TR_CMD register is set, the transmitter will await a new token anyway.
14	Packet available for transmission. In Token Ring operation, the packet could not be sent on the same token as the previous packet.
All other	(Reserved, do not use)

Note For Ethernet operation, the commands 0x10 and 0x14 are equivalent.

The **ENDPTR** field shall contain the ring buffer position of the last byte of the packet.

The **PACKET_DATA** contains all bytes of the packet, up to but not including the FCS, which is appended by the transmitter. Token Ring packets should contain the AC and FC fields.

The Ethernet Preambler and SFD fields are appended by the transmitter, and should not be included in the packet.

Each packet starts at a 256-byte boundary. After the end of the packet, the rest of the current 256-byte block should be skipped. As a special case, if the packet ends at a 256-byte boundary, the following 256-byte block is skipped.

4.3.7 Transmit ring buffer operation.

The start of the packet is set by the **R_TR_START** register. This register sets bit 15-8 of the start address of the packet. Since all packets start at a 256-byte boundary, the lower address bits are all 0.

The transmitter receives a command to start transmitting by reading the **COMMAND** byte of the packet. This can be initiated either by reaching the end of the previous packet, or by the software setting the transmit bit in the **R_TR_CMD** register. The transmitter then reads the **ENDPTR** value and the first data byte, and waits for an opportunity to start transmission (A free Token in the Token Ring case, a quiet line in the Ethernet case). The transmitter maintains the value of the **R_TR_START** register

until the packet is completely transmitted. The **R_TR_START** register can be read by software.

When the transmission of a packet is completed, the **R_TR_START** register is updated to point to the first 256-byte boundary following the packet, and the transmitter reads a COMMAND byte at this position to determine wether there are more packets queued for transmission. When the transmitter sees a 0x00 **COMMAND** byte, it stops, and if the **pkt_intr_en** bit of the **R_TR_CMD** register is set, it issues a Packet Transmitted interrupt. The transmitter can also be forced to stop after the current packet by setting the **transmit** bit of the **R_TR_CMD** bit to 0.

4.3.8 Cancellation of pending transmit packets

It is sometimes necessary to interrupt the transmission of packets already queued in the ring buffer. The transmitter can be forced to stop after the ongoing packet by setting the **transmit** bit in the **R_TR_CMD** register to 0.

To get a more efficient cancellation of any pending transmit packets, also set the **cancel** bit of the **R_TR_CMD** register to 1. This will inhibit all transmission retries for the ongoing packet. Also, if the transmission of the current packet hasn't been started, the internal pipeline will be flushed, and the packet is aborted.

4.3.9 Recovery from transmit errors

If the transmission of a packet fails, and the **retransmit** bit of the **R_TR_CMD** register is set, the packet is automatically retransmitted. After 15 unsuccessfull reties (or immediately, if the retransmit bit is cleared) the transmitter stops and issues an Excessive Retry interrupt. The **R_TR_START** register contains the start position of the packet that failed.

The excessive retry error condition (and the interrupt) is cleared by writing a new value to the **R_TR_START** register.

Note The **transmit** bit of the **R_TR_CMD** register remains set when the transmitter stops due to excessive errors. If the transmit bit isn't cleared by software, the write to the **R_TR_START** register will immediately restart the transmitter operation at the new position.

4.3.10 Transmit ringbuffer interrupts

The two interrupts associated with the transmit ring buffer are the **Packet Transmitted interrupt** and the **Excessive Retry interrupt**.

If the **pkt_intr_en** bit of the **R_TR_CMD** register is set, the Packet Transmitted interrupt is generated each time the transmitter stops after a completed transmission or a cancelled packet. It is also generated if the transmitter is started with the **pkt_intr** bit set, and the buffer is empty. It is not generated if the transmitter stops due to excessive transmit errors. (The Packet Transmitted interrupt will however occur if the

R_TR_CMD register is loaded with a value that sets the **pkt_intr_en** bit and clears the transmit bit, while the transmitter is in the excessive transmit error state.)

The Packet Transmitted interrupt is cleared by setting the **pkt_intr_en** bit of the **R_TR_CMD** register to 0, or by starting the transmission of a new packet.

The Excessive Retry interrupt is generated each time the transmitter stops due to excessive transmit errors. It is also generated when a pending packet is aborted with the **cancel** bit of the **R_TR_CMD** register. The Excessive Retry interrupt is cleared by writing to the **R_TR_START** register, see note in section 4.3.9 above.

4.3.11 Initialization.

At initialization, the software sets the **R_TR_POS** and **R_RT_SIZE** registers, and loads **R_TR_START** register with the upper byte of the start address of the first packet to be sent. When the first packet is ready in the buffer, the software writes the appropriate command in the **R_TR_CMD** register to start the transmission.

5 BUS INTERFACE

The ETRAX bus interface provides an 8/16-bit data bus, a 24-bit address bus and five internally decoded chip select outputs. It also supports two DRAM banks without external logic.

5.1 ADDRESS AND CHIP SELECTS.

The ETRAX chip operates with a 32-bit address space internally, but only 24 address bits are available on the external pins. The upper addresses are decoded to give five different memory chip select outputs, plus RAS/CAS strobes for two DRAM banks. The following chip selects and DRAM controls are provided:

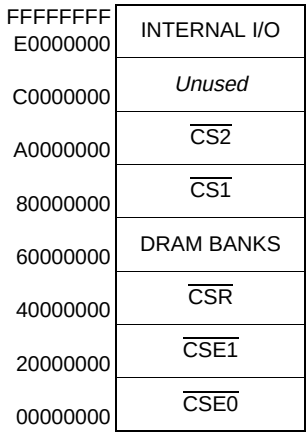
Name:	Description:
$\overline{\text{CSE0}}$	EPROM/flashPROM bank 0 chip select.
$\overline{\text{CSE1}}$	EPROM/flashPROM bank 1 chip select.
$\overline{\text{CSR}}$	SRAM chip select.
$\overline{\text{RAS0}}$	DRAM bank 0 row address strobe.
$\overline{\text{CAS0}}$	DRAM bank 0 column address strobe.
$\overline{\text{RAS1}}$	DRAM bank 1 row address strobe.
$\overline{\text{CAS1}}$	DRAM bank 1 column address strobe.
$\overline{\text{CS1}}$	Peripheral chip select 1.
$\overline{\text{CS2}}$	Peripheral chip select 2.

The $\overline{\text{CSE0}}$, $\overline{\text{CSE1}}$ and $\overline{\text{CSR}}$ are decoded from the internal address only, so accesses to these memory areas have to be qualified by the read or write strobes. The peripheral chip selects $\overline{\text{CS1}}$ and $\overline{\text{CS2}}$ are gated with read/write internally, and can be used directly as strobes to external devices.

For SRAM, EPROM and peripheral areas, the address pins always reflect address bit 23 - 0, giving a maximum address space of 16 Mbyte per memory bank. For DRAM areas, the internal address is multiplexed to give appropriate row and column addresses, see description of DRAM interface below. The maximum available DRAM address space is 32 Mbyte per DRAM bank.

5.1.1 Memory map

The upper three bits of the internal address are decoded, giving the following address map:



5-1 ETRAX memory map.

5.2 DRAM INTERFACE

The DRAM interface supports two banks of DRAM chips without external logic. 8-bit or 16-bit DRAM bus width can be selected. Page mode access is supported, which gives a burst bus speed of 16 MHz at 32 MHz oscillator frequency.

For each access to one of the DRAM banks, the upper addresses are compared with the stored address of the previous access to that DRAM bank. If the upper addresses are equal (page hit), a CAS only cycle is performed. Else (page miss) a complete RAS-CAS cycle is performed, where the row address is output on address bits A0 - A12 during the RAS portion of the cycle. The row address is then stored for next comparison.

The DRAM interface can be configured to eight different modes, to allow for different DRAM types and bus widths:

Mode	Addr. bits checked for page hit/miss	Row addr. output on pins	DRAM chip type examples	Max total address space
0	A[9] - A[21]	A[0] - A[12]	256k×4, 512k×8, 1M×16	8 Mbytes
1	A[10] - A[21]	A[1] - A[12]	256k×4, 256k×16, 4M×4	8 Mbytes
2	A[10] - A[22]	A[0] - A[12]	1M×4, 4M×4	16 Mbytes
3	A[11] - A[22]	A[1] - A[12]	1M×4, 4M×4	16 Mbytes
4	A[11] - A[23]	A[0] - A[12]	4M×4	32 Mbytes
5	A[12] - A[23]	A[1] - A[12]	4M×4	32 Mbytes
6	A[12] - A[24]	A[0] - A[12]	(expected 16Mx4)	64 Mbytes
7	A[13] - A[24]	A[1] - A[12]	(expected 16Mx4)	64 Mbytes

Note: Modes 6 and 7 are based on the assumption that 16 Mbit x 4 DRAM with basically the same spec as 4 M x 4 will be available in the future. The 16 M x 4 type does not exist today.

The DRAM interface supports the CAS-before-RAS refresh method. The refresh interval is set by the **ref_sel** field of the **R_BUS_MODE** register, see page 59. The following refresh intervals are provided:

ref_sel	refresh interval
0	disabled
1	8.7 µs interval
2	13 µs interval
3	52 µs interval

Note: The baudrate timer (see section 3.2.1, page 31) should be started before the DRAM refresh interval timer is started. The above given interval times assumes that the timer input frequency is set to 921.6 kHz, see section 3.1.3, page 30.

For compatibility with existing DRAM types, an initial pause of 100 µs is required after power up before the refresh is enabled. This should be followed by 8 or more CAS-before-RAS refresh cycles before the DRAM is used for data storage.

The bus width, operating mode and number of waitstates for the DRAM interface is controlled by the **R_DRAM_MODE** register, see page 60.

5.3 INTERNAL BUS ARBITRATION

The bus interface performs the bus arbitration between the possible internal bus masters. The bus priority order is:

1. Shared RAM highest priority
2. ETR
3. DRAM refresh
4. DMA
5. CPU lowest priority

The Shared RAM and ETR blocks can only address the SRAM memory area (40000000 - 5FFFFFFF hex), while the CPU (including DMA) can access the total 4 Gbyte address space. Shared RAM and ETR cycles can be run in the middle of a DRAM RAS-CAS or refresh cycle, giving a short bus latency time for these units.

5.4 BUS STATUS

The bus interface provides three bus status outputs BS<2> - BS<0>.

Bus status	Bus cycle type
0	(unused)
1	CPU code access
2	CPU data access
3	CPU INTA sequence
4	DMA channel 0
5	DMA channel 1
6	ETR access
7	Shared RAM or idle cycle

Instruction fetch cycles are classified as code access, while all data read/write cycles are classified as data access, including immediate values and absolute addresses. CPU INTA sequence includes the interrupt vector number fetch and the subsequent read of the interrupt vector.

5.5 BUS WIDTH AND WAITSTATES

ETRAX can be configured for 8-bit or 16-bit bus width. The bus width and number of waitstates can be configured individually for the EPROM, SRAM and DRAM banks. The peripheral banks and internal I/O are always 8 bit wide, but the number of waitstates is configurable.

The EPROM bus width (memory area controlled by $\overline{CSE0}$ and $\overline{CSE1}$), and the SRAM bus width (memory area controlled by $\overline{CSR}$), are controlled by the input pins EPROM_W and SRAM_W. The DRAM bus width is configured by software.

The waitstates can be set as follows:	**Waitstate range:**
Early waitstates (inserted before read/write strobes)	0 – 3
Specific waitstates:	
EPROM read cycles ($\overline{CSE0}$ or $\overline{CSE1}$)	0 – 3
SRAM read cycles ($\overline{CSR}$)	0 – 3
EPROM/SRAM write cycles ($\overline{CSE0}$, $\overline{CSE1}$ or $\overline{CSR}$)	0 – 3
DRAM CAS (page hit) cycles ($\overline{CAS0}$ or $\overline{CAS1}$) ***(see note)***	0 – 1
DRAM RAS precharge (page miss) cycle	0 – 1
DRAM RAS cycle	0 – 1
DRAM CAS-before-RAS refresh cycle	1 – 2
Peripheral ($\overline{CS1}$ or $\overline{CS2}$), internal I/O, and INTA cycles. (also controls DMA to external I/O)	0 – 3

Note: For a DRAM write cycle, one extra waitstate is inserted if the previous bus cycle was a memory read. This is necessary to give enough turn off time to the memories.

Early waitstates are controlled by the **R_EARLY_WS** register, and the specific waitstates are controlled by the **R_LATE_WS** register, see page 58.

An external wait pin ($\overline{WAIT}$) is provided, which can be used by external devices to insert extra waitstates. When used, the $\overline{WAIT}$ input should be pulled low immediately after $\overline{RD}$, $\overline{WRH}$ or $\overline{WRL}$ goes low, and must be kept low for at least one system clock cycle. The low-to-high transition of $\overline{WAIT}$ need not be synchronized to the system clock, as long as the minimum time of one clock cycle from $\overline{RD}$, $\overline{WRH}$ or $\overline{WRL}$ low to $\overline{WAIT}$ high is satisfied. The use of the $\overline{WAIT}$ input also requires that the number of specific waitstates for the memory area in question is set to 1 or more.

A zero waitstate bus cycle takes 2 oscillator clock cycles. The total number of clock cycles for a bus cycle is given by

```
2 + <number of early waitstates> + <number of specific waitstates>
```

Default after reset is for the DRAM area 8-bit bus and minimum number of waitstates. Default after reset for all other memory areas and bus cycle types is maximum number of waitstates.

5.6 POWER SAVE MODE

The execution speed of the internal CPU can be reduced by up to 16 times for lower power consumtion. The speed reduction is controlled by the **cpu_interval** field of the **R_BUS_MODE** register. The power save mode affects the CPU and DMA operations, but has no effect on the Ethernet/Token Ring or Shared RAM cycles.

5.7 NORMAL AND EXTENDED WRITE MODE

During a normal ETRAX write cycle, the $\overline{\text{WRH}}$ and/or $\overline{\text{WRL}}$ strobes will go high one half clock cycle before the end of the bus cycle. The write pulse can be extended to the end of the bus cycle by setting the **wrm** bit in the **R_BUS_MODE** register, see page 59. Write cycles to the DRAM areas will always use the extended write mode, regardless of the wrm bit.

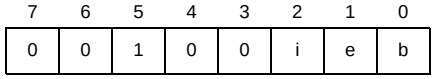
6 INTERRUPT CONTROL

The interrupt control block handles the interrupts to the internal CPU. Interrupts can be generated either by internal logic or externally through the $\overline{\text{IRQ}}$ pin.

External interrupts can either use the internally generated interrupt vector number, or supply an interrupt vector number on the data bus. External interrupts with external vector number are acknowledged on the $\overline{\text{INTA}}$ output pin. $\overline{\text{INTA}}$ is inactive for interrupt cycles with internally generated vector number. Internally generated interrupts have priority over external interrupts with external vector numbers.

6.1 INTERRUPTS WITH INTERNAL VECTOR NUMBER

For internal interrupts and external interrupts with internal vector number, the interrupt vector number is generated as:



Receive/transmit ring buffer interrupts:
If any of the four ring buffer handler interrupts are generated, this bit in the vector number is set.

Network interface interrupts:
If any of the eigtht network interface interrupts are generated, this bit in the interrupt vector number is set.

I/O port interrupts:
If any of the seven internal I/O interrupts or an external interrupt with internal vector number is generated, this bit in the interrupt vector number is set.

There are three interrupt status registers, that show which of the interrupts that are generated within each group. Interrupt sources within each group of interrupts can be masked using the interrupt mask registers. ETRAX has separate register addresses for the set mask and clear mask operations. This allows independent manipulation of the individual bits in the interrupt masks. The registers associated with the interrupt masks are shown in the table below:

Group:	**Set mask**	**Clear mask**	**Read status**
Receive/transmit ring buffer:	R_BUF_MASKS	R_BUF_MASKC	R_BUF_STATUS
Network interface interrupts:	R_ETR_MASKS	R_ETR_MASKC	R_ETR_STATUS
I/O port interrupts:	R_IO_MASKS	R_IO_MASKC	R_IO_STATUS

6.2 INTERRUPTS WITH EXTERNAL INTERRUPT VECTOR

External interrupts with externally supplied vector numbers are enabled/disabled with the mask registers **R_EXT_MASKS** and **R_EXT_MASKC**. When an external vector number is used, the external interrupt bit in the I/O port interrupt mask must be cleared.

An external interrupt with externally supplied vector number is acknowledged by the $\overline{\text{INTA}}$ output going low during the interrupt acknowledge cycle. The external device should supply the vector number on the lower 8-bits of the data bus in response to the $\overline{\text{INTA}}$ signal.

6.3 INTERRUPT ACKNOWLEDGE SEQUENCE

The interrupt acknowledge sequence consists of the following steps:

1. Perform an INTA cycle, where the 8-bit vector number is read from the bus.
2. Store the contents of PC to the Interrupt Return Pointer (IRP). Note that the return address is not automatically pushed on the stack.
3. Read the interrupt vector from the address [IBR + <vector number> * 4].
4. Start the execution at the address pointed to by the interrupt vector.

Only bit 29 and 30 of the Interrupt Base Register (IBR) are actually implemented. The remaining bits are always zero. See also section 1.5, page 16.

6.4 SPURIOUS INTERRUPTS

Spurious interrupts may occur if the interrupt request signal from an internal or external source is removed at the same time as the internal logic recognizes the interrupt. If this happens, the spurious interrupt vector number 32 (20 hex) is generated.

6.5 INITIALIZATION

All interrupt masks are undefined after reset. In order to avoid unwanted interrupts, all interrupt mask bits must be initialized before the interrupts are enabled in the CPU.

7 REGISTER DEFINITIONS

7.1.1 Base address for ETR mode registers

ETRAX_IOBASE = 0xE0000000

All register addresses in this document are relative to ETRAX_IOBASE and are given in hexadecimal format.

7.1.2 Bus IF: 00 - 07

Address:	read/write:	Internal I/O signals:	Initial value:
00	write	R_LATE_WS<7:0>	FF
01	write	R_EARLY_WS<1:0>	03
02	write	R_BUS_MODE<7:0>	00
04	write	R_DRAM_MODE<7:0>	00
06	write	R_CLOCK_MODE<7:0>	FF
07	write	R_DMA_CONFIG<7:0>	00

7.1.3 Interrupt mask: 10 - 17

Address:	read/write:	Internal I/O signals:	Initial value:
10	read	R_IO_STATUS<7:0>	n.a.
10	write	R_IO_MASKS<7:0>	Undefined
11	write	R_IO_MASKC<7:0>	Undefined
12	read	R_ETR_STATUS<7:0>	n.a.
12	write	R_ETR_MASKS<7:0>	Undefined
13	write	R_ETR_MASKC<7:0>	Undefined
14	read	R_BUF_STATUS<3:0>	n.a.
14	write	R_BUF_MASKS<3:0>	Undefined
15	write	R_BUF_MASKC<3:0>	Undefined
16	write	R_EXT_MASKS	Undefined
17	write	R_EXT_MASKC	Undefined

7.1.4 Shared Ram: 28 - 29

Address:	read/write:	Internal I/O signals:	Initial value:
28	write	R_MIO_ADDR<7:0>	Undefined
29	write	R_MIO_MODE<2:0>	00

7.1.5 General Purpose I/O Port: 30 - 31

Address:	read/write:	Internal I/O signals:	Initial value:
30	read	R_P1_DATA_IN<7:0>	n.a.
30	write	R_P1_DATA_OUT<7:0>	Undefined
31	write	R_P1_DIR<7:0>	00

7.1.6 Port2: 34 - 35

Address:	read/write:	Internal I/O signals:	Initial value:
34	read	R_P2_DATA<3:0>	n.a.
34	write	R_SER_CONNECT<7:0>	FF
35	write	R_P2_CONFIG<1:0>	00

7.1.7 Port 3: 38

Address:	read/write:	Internal I/O signals:	Initial value:
38	read	R_P3_DATA<6:0>	n.a.

7.1.8 Par. 1: 40 - 45

Address:	read/write:	Internal I/O signals:	Initial value:
40	read	R_PAR_BACKCH<7:0>	n.a.
40	write	R_PAR_DATA<7:0>	Undefined
41	read	R_PAR_STAT<7:0>	n.a.
41	write	R_PAR_DSETUP<4:0>	Undefined
42	write	R_PAR_STROBE<4:0>	Undefined
43	write	R_PAR_DHOLD<4:0>	Undefined
44	write	R_PAR_CTRL<4:0>	04
45	write	R_PAR_CONFIG<5:0>	00

7.1.9 Par. 2: 48 - 4D

Address:	read/write:	Internal I/O signals:	Initial value:
48	read	R_PCH_DATA<7:0>	n.a.
48	write	R_PAR2_DATA<7:0>/ R_PCH_BACKCH<7:0>	Undefined
49	read	R_PCH_STAT<7:0>/ R_PAR2_STAT<7:0>	n.a.
49	write	R_PCH_ACKTIME<3:0>/ R_PAR2_STROBE<4:0>	0F
4A	write	R_PAR2_DHOLD<4:0>	Undefined
4B	read	R_PAR2_BACKCH<7:0>	n.a.
4B	write	R_PAR2_DSETUP<4:0>	Undefined
4C	write	R_PCH_CTRL<5:0>/ R_PAR2_CTRL<4:0>	0C
4D	write	R_PCH_CONFIG<5:0>/ R_PAR2_CONFIG<5:0>	00

7.1.10 Serial I/O Registers: 50 - 53

Address:	read/write:	Internal I/O signals:	Initial value:
50	read	R_SER1_DIN<7:0>	n.a.
50	write	R_SER1_DOUT<7:0>	Undefined
51	read	R_SER1_STAT<6:0>	n.a.
51	write	R_SER1_BAUD<7:0>	Undefined
52	write	R_SER1_IN_CFG<6:0>	00
53	write	R_SER1_OUT_CFG<7:0>	00

7.1.11 Timer: 58 - 5A

Address	read/write:	Internal I/O signals:	Initial value:
58	write	R_TIMER_DATA<7:0>	Undefined
59	write	R_TIMER_MODE<7:0>	00
5A	write	R_TIMER_INTA	n.a

7.1.12 ETR: 80 - 94

Address	read/write:	Internal I/O signals:	Initial value:
80	write	R_MA0<7:0>	Undefined
81	write	R_MA1<7:0>	Undefined
82	write	R_MA2<7:0>	Undefined
83	write	R_MA3<7:0>	Undefined
84	write	R_MA4<7:0>	Undefined
85	write	R_MA5<7:0>	Undefined
88	write	R_GA0<7:0>	Undefined
89	write	R_GA1<7:0>	Undefined
8A	write	R_GA2<7:0>	Undefined
8B	write	R_GA3<7:0>	Undefined
8C	write	R_GA4<7:0>	Undefined
8D	write	R_GA5<7:0>	Undefined
8E	write	R_GA6<7:0>	Undefined
8F	write	R_GA7<7:0>	Undefined
90	write	R_TR_MODE1<7:0>	Undefined
91	write	R_TR_MODE2<4:0>	00
92	write	R_TVX<5:0>	Undefined
93	write	R_TRR<7:0>	Undefined
94	write	R_ANALOG<3:0>	Undefined

7.1.13 Packet receive/transmit: 98 - 9E

Address	read/write:	Internal I/O signals:	Initial value:
98	read	R_REC_STATUS<4:0>	n.a.
98	write	R_REC_MODE<2:0>	01
99	write	R_REC_END<7:0>	Undefined
9A	write	R_REC_POS<7:0>	Undefined
9B	write	R_RT_SIZE<3:0>	Undefined
9C	read	R_TR_STATUS<7:0>	n.a.
9C	write	R_TR_CMD<5:0>	01
9D	read	R_TR_START<7:0>	n.a.
9D	write	R_TR_START<7:0>	Undefined
9E	write	R_TR_POS<7:0>	Undefined

7.2 BIT DELAY

Some bits in the R_TR_MODE1, R_TR_MODE2, R_TRR and R_ANALOG registers require that time between changes to the register is at least 8 or 16 bit-times. This requirement is necessary because of the synchronization mechanism between the CPU clock region and the network interface clock region.

To guarantee that the requirement is met you have to take into account the current transfer rate on the network (4/10/16 Mbit/s), cpu clock frequency, bus width and waitstates in memory accesses by the CPU. Note that the same program code should work for both 4 & 16-Mbit/s token-ring.

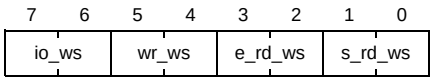
One solution that is guaranteed to always work is to use the maximum required delay of all parameter combinations. The maximum delay required occurs with 4Mbit/s, 16-bit bus width, zero waitstates and 16 MHz cpu clock. An 8 bit times delay should then take 16/4*8 = 32 CPU cycles or 32 NOP instructions. The example below shows a more code effective solution than 32 NOPs. Improving this code with respect to code size and exact delay is left as an exercise to the reader.

```
;
; Guaranteed at least 8 bit times delay (34 CPU cycles).
;
MOVE.D 0xC0000000, R0  ; Point to unused area in memory map.
(3 cycles)
MOVEM  SP, [R0] ; Write R0-R14 to unused memory area.
(15*2+1 cycles)
```

7.3 BUS INTERFACE MODE REGISTERS

R_LATE_WS<7:0> (write)

Address: 0x00	**Initial Value:** 0xFF



SRAM read waitstates

EPROM read waitstates

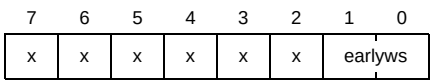
SRAM/EPROM write waitstates

Waitstates for internal I/O, $\overline{CS1}$, $\overline{CS2}$, INTA cycles and external DMA cycles.

The wait state fields are coded as:
00 - 0 wait states
01 - 1 wait states
10 - 2 wait states
11 - 3 wait states

R_EARLY_WS<1:0> (write)

Address: 0x01	**Initial Value:** 0x03

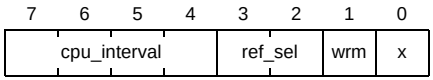


Early waitstates, inserted before $\overline{RD}$, $\overline{WR}$ or $\overline{INTA}$.

The wait state fields are coded as:
00 - 0 wait states
01 - 1 wait states
10 - 2 wait states
11 - 3 wait states

R_BUS_MODE<7:0> (write)

Address: 0x02	**Initial Value:** 0x00



(Not used)

Write mode. Sets the write pulse to end at the end of the bus cycle, or one half clock cycle earlier. Does not affect write to DRAM:s.
0: Early end of write.
1: Late end of write.

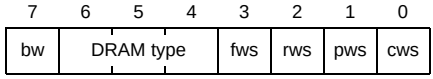
DRAM refresh interval select: ***see Note***
0: Refresh disabled.
1: 8.7 µs interval.
2: 13 µs interval.
3: 52 µs interval.

CPU slow down factor for power save. Sets number of idle cycles to insert between each CPU or DMA cycle. Does not affect shared RAM or Ethernet/Token Ring cycles.

Note: The baudrate generator must be enabled and the dram_mode register must be initialized before enabling the DRAM refresh. In order to meet DRAM specifications, wait at least 100µs after reset before enabling the refresh, and then wait at least 8 times the refresh interval before using the DRAM.

R_DRAM_MODE<7:0> (write)

Address:	0x04	**Initial Value:**	0x00



CAS cycle waitstate: ***see Note***
0: 0 waitstates.
1: 1 waitstate.

RAS precharge waitstate.Inserted after a RAS low-to-high transition.
0: 0 waitstates.
1: 1 waitstate.

RAS cycle waitstate. Inserted after a RAS high-to-low transition, except during refresh.
0: 0 waitstates.
1: 1 waitstate.

Refresh waitstates. Inserted after RAS high-to-low in a refresh cycle.
0: 1 waitstate.
1: 2 waitstates.

DRAM type:

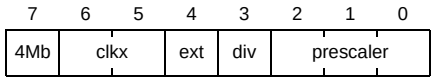
value	**type:**	**Row addr:**	**Row addr pins:**
000	0:	A9 - A21	A0 - A12
001	1:	A10 - A21	A1 - A12
010	2:	A10 - A22	A0 - A12
011	3:	A11 - A22	A1 - A12
100	4:	A11 - A23	A0 - A12
101	5:	A12 - A23	A1 - A12
110	6:	A12 - A24	A0 - A12
111	7:	A13 - A24	A1 - A12

DRAM bus width:
0: 8-bit wide DRAM banks
1: 16-bit wide DRAM banks

Note: One extra waitstate is added for a write cycle following a memory read or external INTA cycle. Also one extra wait is added for DMA transfers from an external I/O.

R_CLOCK_MODE<7:0> (write)

Address: 0x06	**Initial Value:** 0xFF



Prescaler for timers. Should be set to match system oscillator frequency.
0: 20.0 / 26.6 / 33.3 MHZ
1: 18.0 / 24.0 / 30.0 MHz
2: 18.7 / 25.0 / 31.2 MHz
3: 19.2 / 25.6 / 32.0 MHz
4: Sync. mode ***see Note***
5: -
6: -
7: Stopped

Divider for Ethernet/Token ring transmit clock.
0: Divide by 4. (4 Mb Tr)
1: Divide by 1. (16 Mb Tr / Ethernet)

External/internal Ethernet/Token ring transmit clock.
0: Internal (Token ring)
1: External (Ethernet)

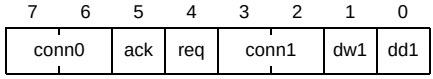
Peripheral clock divide. Should be set to match system oscillator frequency.
0: Divide by 3. (18 - 20 MHz clock)
1: Divide by 4.(24 - 26.6 MHz clock)
2: Divide by 5.(30 - 33.3 MHz clock)
3: Divide by 6.(Not useful.)

Token ring 4 Mb/16 Mb select. (Only affects token hold time.)
0: 16 Mb Token ring.
1: 4 Mb Token ring.

Note: The prescale timer has to be set to the sync mode for at least 50 clock cycles during startup to give a predictable behavior in simulation and test. Not needed for normal operation.

R_DMA_CONFIG<7:0> (write)

Address: 0x07	**Initial Value:** 0x00



DMA channel 1 direction: (Must match selected connection)
0: In (write to memory)
1: Out (read from memory)

DMA channel 1 data width:
0: 8-bit transfers
1: 16-bit transfers (Only allowed for external I/O)

DMA channel 1 connection:
0: Disabled
1: Parallel port 1 out
2: Serial out
3: External I/O

External DMA request polarity:
0: Active low
1: Active high

External DMA acknowledge polarity:
0: Active low
1. Active high

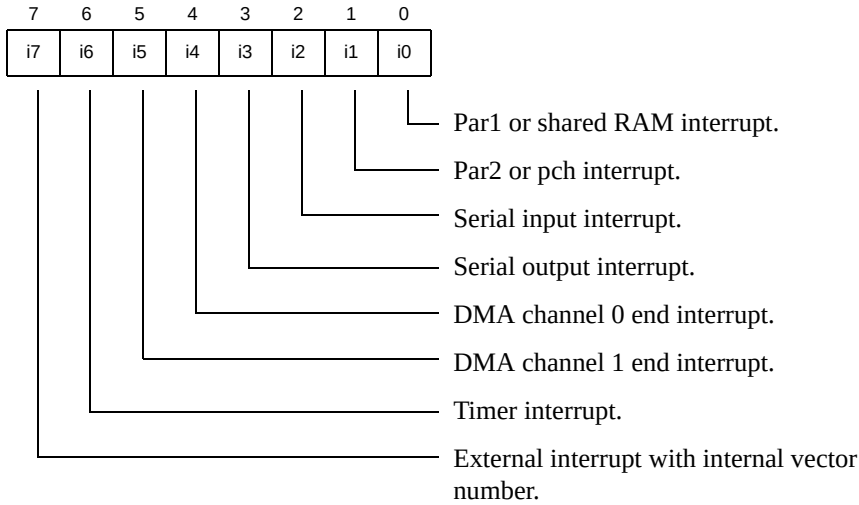
DMA channel 0 connection:
0: Parallel input
1: Parallel port 2 out
2: (illegal)
3: Parallel port 1 out

7.4 INTERRUPT STATUS AND MASK REGISTERS

R_IO_STATUS<7:0> (read)

Address:	0x10	**Initial Value:**	N/A

All internal I/O interrupts are OR:ed together to generate bit 2 in the internally generated interrupt vector number. The R_IO_STATUS register indicates the interrupt source that generated the interrupt. 1's represent active interrupts.



These status bits are cleared individually.

The shared RAM interrupt is cleared by a 0 to 1 transition on the 'clr_intr' bit in the R_MIO_MODE register.

In input mode the par1 interrupt is cleared by reading a byte from the R_PAR_BACKCH register, and in output mode the par1 interrupt is cleared by writing a byte to the R_PAR_DATA register. After the last byte is sent in output mode the par1 interrupt has to be masked.

In input mode the par2 interrupt is cleared by reading a byte from the R_PAR2_BACKCH register, and in output mode the par2 interrupt is cleared by writing a byte to the R_PAR2_DATA register. After the last byte is sent in output mode the par2 interrupt has to be masked.

In input mode the pch interrupt is cleared by reading a byte from the R_PCH_DATA register, and in output mode the pch interrupt is cleared by writing a byte to the R_PCH_BACKCH register. After the last byte is sent in output mode the pch interrupt has to be masked.

The serial input interrupt is cleared by reading a byte from the R_SER1_DIN register.

The serial output interrupt is cleared by writing a byte to the R_SER1_DOUT register. When the last byte is sent the serial output interrupt has to be masked.

The DMA ch0 and DMA ch1 interrupts can't be cleared, but has to be masked when the DMA transfer is ready. DMA is started by setting the D- or E-flags in the CPU CCR register. When the DMA transfer is ready these flags are cleared, and the interrupt bits set. The interrupts bits can also be read in R_P3_DATA register as the dma0_end and dma1_end bits.

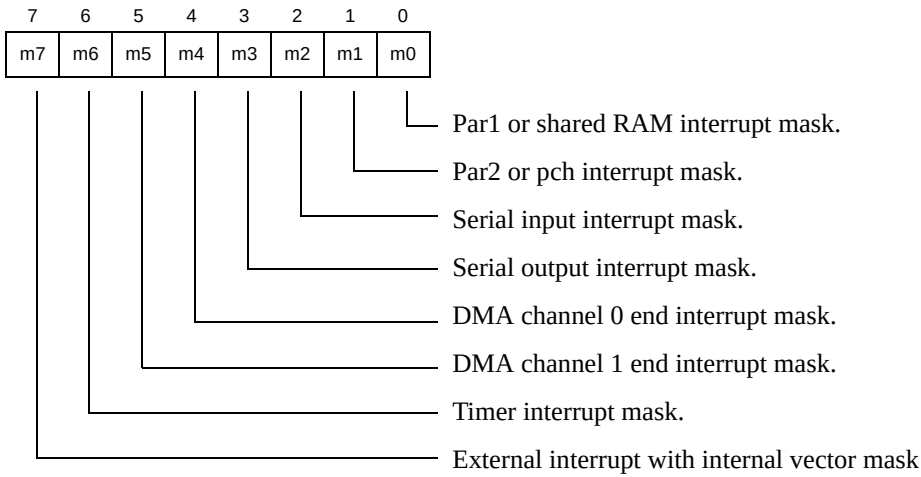
The timer interrupt is cleared by writing a dummy byte to the R_TIMER_INTA register.

The external interrupt is cleared by clearing the external interrupt source

R_IO_MASKS<7:0> (write)

Address:	0x10	**Initial Value:**	Undefined

This port can set the individual mask bits in the interrupt mask. Mask bits corresponding to 0's in the data written are not affected. (they will keep their previous value). Interrupts are enabled if the mask bits are set to 1.

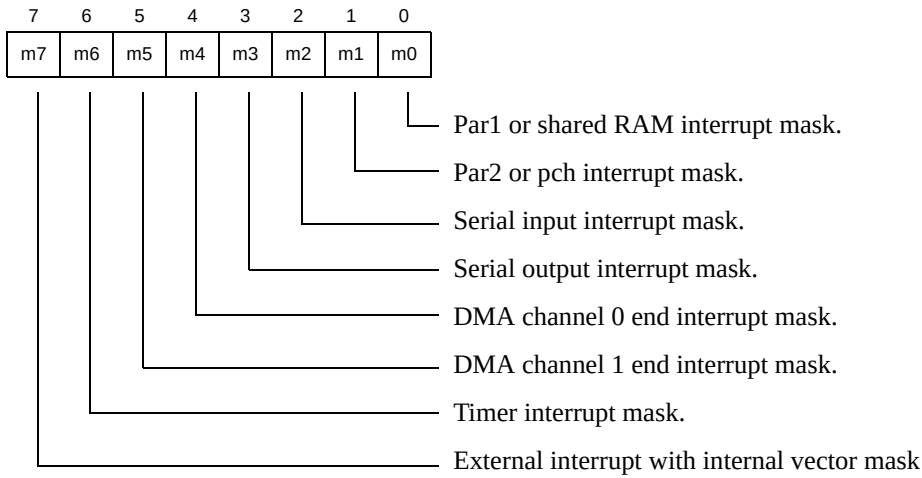


R_IO_MASKC<7:0> (write)

Address:	0x11	**Initial Value:**	–
		Recommended setting:	–
		Startup:	0xFF

This port can clear the individual mask bits in the interrupt mask. Mask bits corresponding to 0's in the data written are not affected. (they will keep their previous value). Interrupts are disabled if the mask bits are set to 1.

This port should be written with FF before the interrupts in the CRIS cpu are enabled, to clear the interrupt mask.



R_ETR_STATUS<7:0> (read)

Address:	0x12	**Initial Value:**	N/A

All Token Ring interrupts are OR:ed together to generate bit 1 in the internally generated interrupt vector number. The R_ETR_STATUS register indicates the interrupt source that generated the interrupt. 1's represent active interrupts..

To read and clear R_ETR_STATUS the following procedure is necessary:

1. Set rd_intr_status (R_TR_MODE2<0>) to one (1).

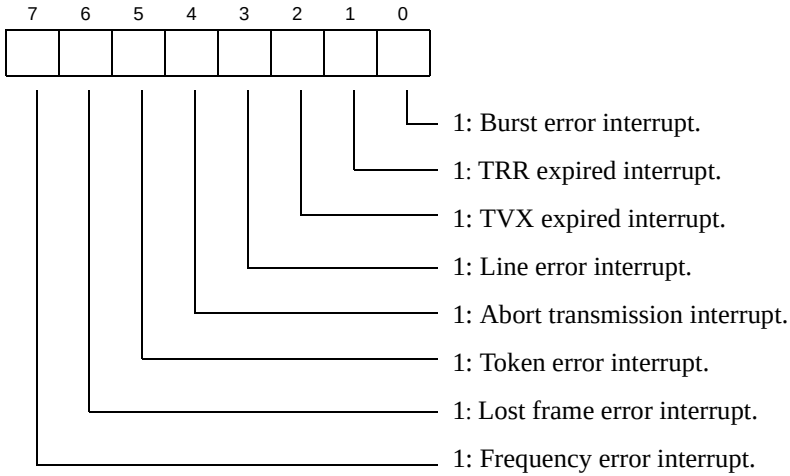
This updates R_ETR_STATUS and disables any further ethernet/token-ring interrupts. While the etr interrupts are disabled they are still latched in the hardware and if there was an etr interrupt during the disable time a new interrupt will be generated immediately when rd_intr_status is cleared (see step 4).

2. Wait at least 16 bit times (NETWORK bit times) for internal hardware synchronization.

3. Read R_ETR_STATUS.

4. Clear rd_intr_status (R_TR_MODE2<0>) to zero (0).

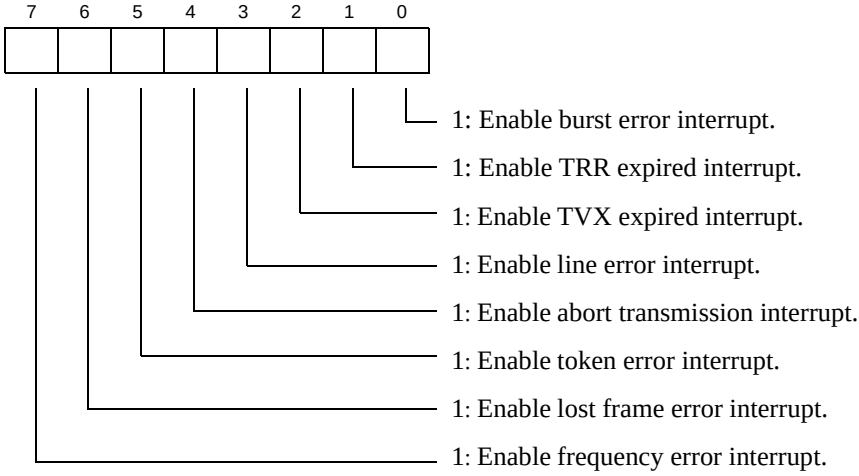
5. It will take up to 16 bit times (NETWORK bit times) before the interrupts in ETR_STATUS are cleared. Therefore it is necessary to wait 16 bit times before interrupts are enabled.



R_ETR_MASKS<7:0> (write)

Address:	0x12	**Initial Value:**	Undefined

This port can set the individual mask bits in the interrupt mask. Mask bits corresponding to 0's in the data written are not affected. (they will keep their previous value). Interrupts are enabled if the mask bits are set to 1.

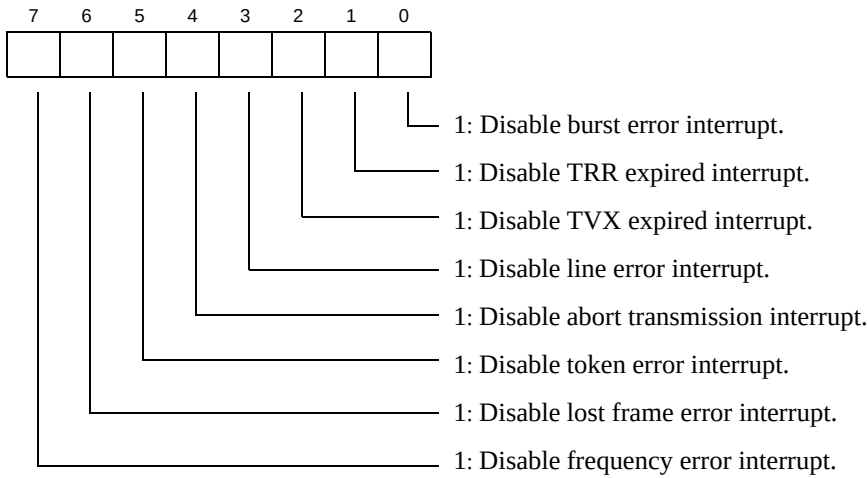


R_ETR_MASKC<7:0> (write)

Address:	0x13	**Initial Value:**	Undefined
		Recommended setting:	–
		Startup:	0xFF

This port can clear the individual mask bits in the interrupt mask. Mask bits corresponding to 0's in the data written are not affected. (they will keep their previous value). Interrupts are disabled if the mask bits are set to 1.

This port should be written with FF before the interrupts in the CRIS cpu are enabled, to clear the interrupt mask.

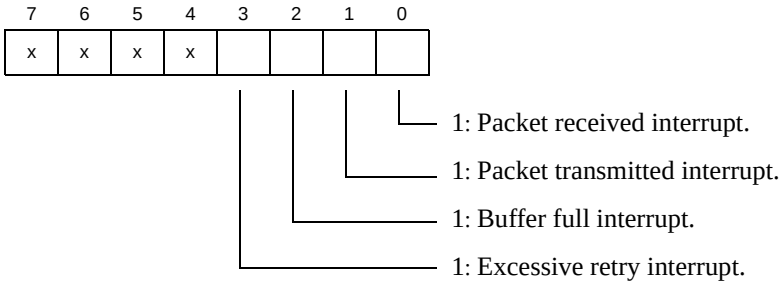


R_BUF_STATUS<3:0> (read)

Address:	0x14	**Initial Value:**	N/A

All ring buffer interrupts are OR:ed together to generate bit 0 in the internally generated interrupt vector number. The R_BUF_STATUS register indicates the interrupt source that generated the interrupt. 1's represent active interrupts.

These status bits are cleared individually. Packet received and buffer full interrupts are cleared by clr_pkt_recvd and clr_buf_full bits in the R_REC_MODE register. Packet transmitted interrupt is cleared by starting transmission or by clearing pkt_intr_en in R_TR_CMD. Excessive retry interrupt is cleared by writing to R_TR_START.



Note: In ethernet mode Execessive retry interrupt signals that there have been 15 transmission failures and that transmission is aborted.

Excessive retry interrupt together with Packet transmitted interrupt signals that the transmission was cancelled due to a SW cancel (cancel bit in R_TR_CMD).

In token ring mode Excessive retry interrupt can never appear alone, but only together with Packet transmitted interrupt or Token error interrupt (R_ETR_STATUS).

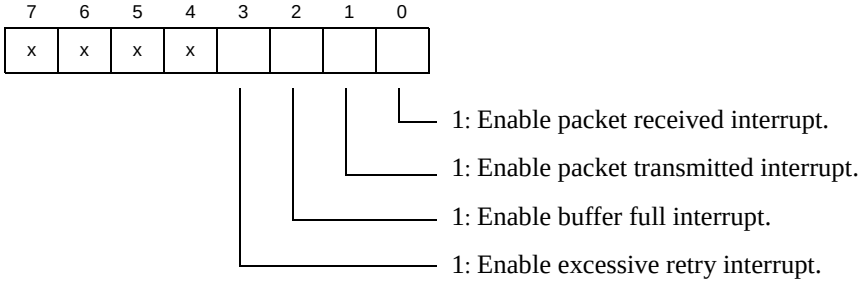
Excessive retry interrupt together with Token error interrupt signals that the transmission was cancelled due to a token error.

When a transmission is aborted and the Excessive retry interrupt generated R_TR_START point to the start of the aborted packet.

R_BUF_MASKS<3:0> (write)

Address:	0x14	**Initial Value:**	Undefined

This port can set the individual mask bits in the interrupt mask. Mask bits corresponding to 0's in the data written are not affected. (they will keep their previous value). Interrupts are enabled if the mask bits are set to 1.

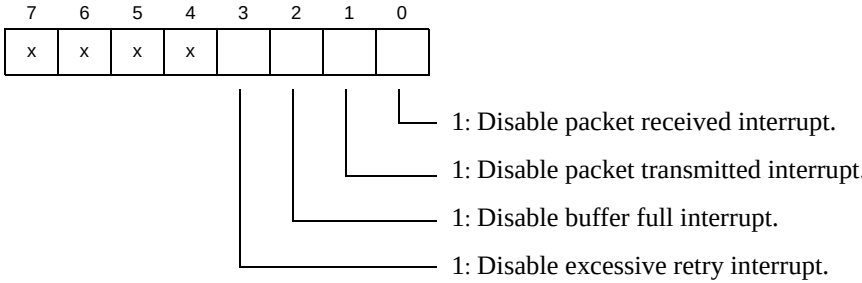


R_BUF_MASKC<3:0> (write)

Address:	0x15	**Initial Value:**	Undefined
		Recommended setting:	–
		Startup:	0xFF

This port can clear the individual mask bits in the interrupt mask. Mask bits corresponding to 0's in the data written are not affected. (they will keep their previous value). Interrupts are disabled if the mask bits are set to 1.

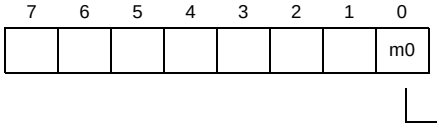
This port should be written with FF before the interrupts in the CRIS cpu are enabled, to clear the interrupt mask.



R_EXT_MASKS<0> (write)

Address:	0x16	**Initial Value:**	Undefined

This port can set the mask bit in the external interrupt mask. If the mask bit is 0 in the data written is not affected (will keep it's previous value). External interrupts are enabled if the mask bit is set to 1.

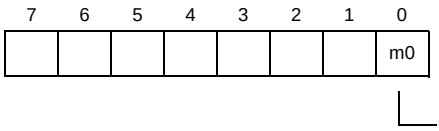


R_EXT_MASKC<0> (write)

Address:	0x17	**Initial Value:**	Undefined
		Recommended setting:	–
		Startup:	0x01

This port can clear the mask bit in the external interrupt mask. If the mask bit is 0 in the data written is not affected (will keep it's previous value). External interrupts are disabled if the mask bit is set to 1.

This port should be written with 0x01 before the interrupts in the CRIS cpu are enabled, to clear the interrupt mask.

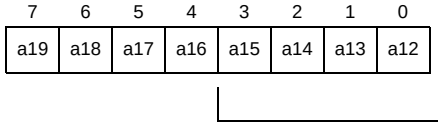


7.5 SHARED RAM INTERFACE REGISTERS

R_MIO_ADDR<7:0> (write):

Address:	0x28	**Initial Value:**	Undefined
		Recommended setting:	–

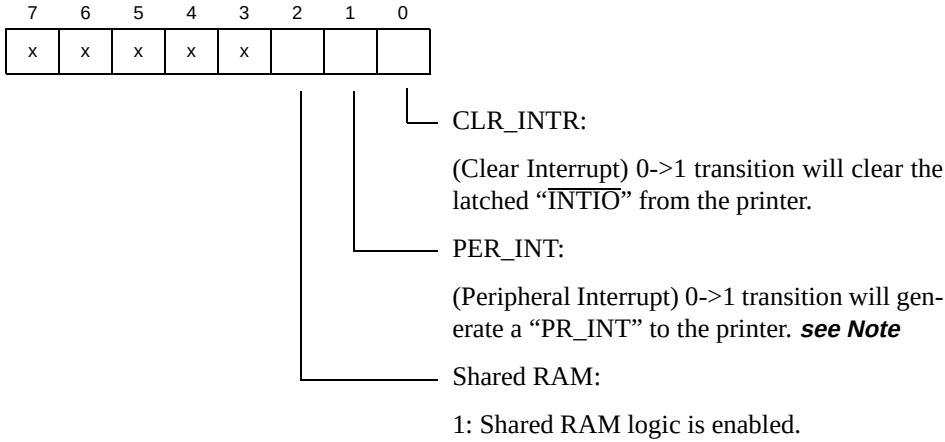
This register controls address bit 19:12 of the memory address during shared RAM access. Address bit 20 and up is always 0 during shared RAM accesses.



R_MIO_MODE<7:0> (read):

Address:	0x29	**Initial Value:**	00
		Recommended setting:	–
		Parallel out::	0x00
		shared RAM interface:	0x04

This register selects shared RAM or parallel output port, and controls the shared RAM interrupts to and from the printer.



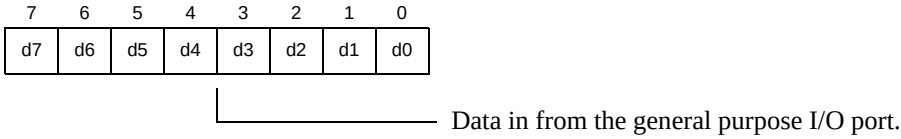
Note: No extra pulses should be sent out before the timeout of 600 - 750 ns has elapsed.

7.6 GENERAL PURPOSE I/O PORT REGISTERS

R_P1_DATA_IN<7:0> (read):

Address:	0x30	**Initial Value:**	N/A

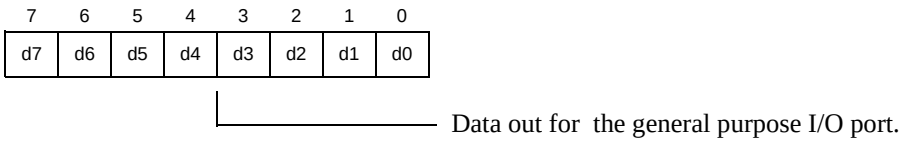
This port shows the input data on the general purpose I/O port.



R_P1_DATA_OUT<7:0> (write):

Address:	0x30	**Initial Value:**	Undefined

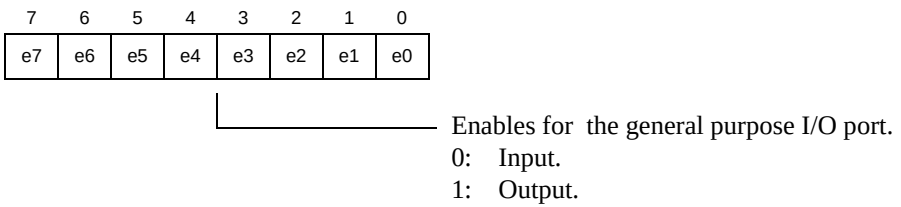
This is the data output for the general purpose I/O port, when used as normal output port.



R_P1_DIR<7:0> (write):

Address:	0x31	**Initial Value:**	0x00

This is the output enables for the general purpose I/O port, when used as normal output port. 0's means inputs, and 1's means outputs.

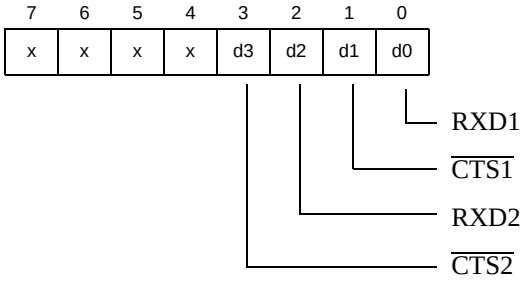


7.7 PORT2 REGISTERS

R_P2_DATA<3:0> (read):

Address: 0x34	**Initial Value:** N/A

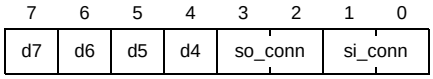
This port shows the input data on port 2.



R_SER_CONNECT<7:0> (write):

Address: 0x34	**Initial Value:** 0xFF
Recommended setting:	0xFA

This register controls the connection of the serial port (port 2) and can also control the output port pins directly.



Connection of the serial input ports, i.e. pins RXD1, $\overline{\text{RTS1}}$, RXD2 and $\overline{\text{RTS2}}$. See table below.

Connection of the serial output ports, i.e. pins TXD1, $\overline{\text{CTS1}}$, TXD2 and $\overline{\text{CTS2}}$. See table below.

Used for controlling the different bits in port 2 individually.

ser_connect<1:0>	**RXD1 (P2<0>)**	**RTS1_ (P2<5>)**	**RXD2 (P2<2>)**	**RTS2_ (P2<7>)**
00	serin RXD	serin $\overline{\text{RTS}}$	---	serin $\overline{\text{RTS}}$
01	---	connect<5>	serin RXD	serin $\overline{\text{RTS}}$
10	serin RXD	serin $\overline{\text{RTS}}$	---	connect<7>
11	---	connect<5>	---	connect<7>

ser_connect<3:2>	**TXD1 (P2<4>)**	**CTS1_ (P2<1>)**	**TXD2 (P2<6>)**	**CTS2_ (P2<3>)**
00	serout TXD	serout $\overline{\text{CTS}}$	serout TXD	---
01	connect<4>	---	serout TXD	serout $\overline{\text{CTS}}$
10	serout TXD	serout $\overline{\text{CTS}}$	connect<6>	---
11	connect<4>	---	connect<6>	---

Note: serin refers to the serial input interface, serout refers to the serial output interface, and connect refers to the ser_connect register.

Do not enable the serial ports before the configuration registers (ser1_in_cfg, ser1_out_cfg) have been set up.

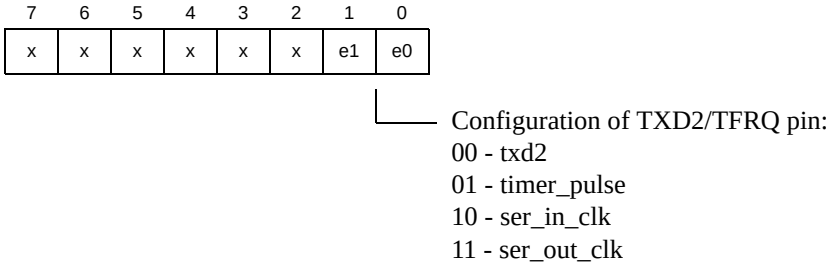
Examples:

ser_connect	**Mode**
FA	Serial port on pins RXD1, $\overline{\text{RTS1}}$, TXD1, $\overline{\text{CTS1}}$, general purpose port on other pins.
F5	Serial port on pins RXD2, $\overline{\text{RTS2}}$, TXD2, $\overline{\text{CTS2}}$, general purpose port on other pins.

R_P2_CONFIG<7:0> (write):

Address:	0x35	Initial Value:	0x00

Configuration of port 2, TXD2/TFRQ pin.

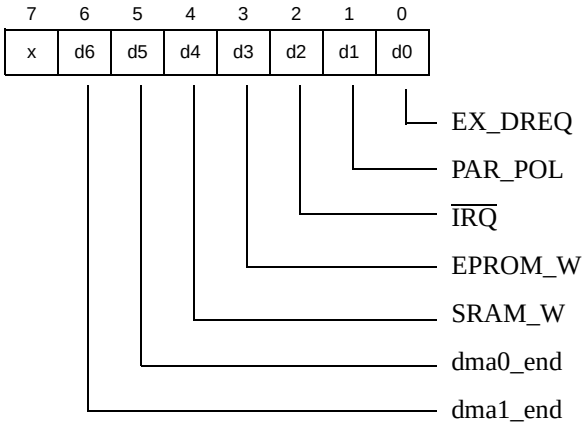


7.8 PORT3 REGISTERS

R_P3_DATA<6:0> (read):

Address:	0x38	Initial Value:	N/A

This port shows some of the package pins and the DMA end count flags. $\overline{IRQ}$ is internally synchronized.



Note: dma0_end shows the inverse of the CPU E-flag.

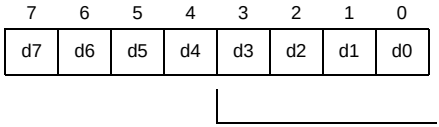
dma1_end shows the inverse of the CPU D-flag.

7.9 PARALLEL PORT1 REGISTERS

R_PAR_BACKCH<7:0> (read):

Address:	0x40	**Initial Value:**	N/A

This register contains data read from the parallel output interface when in reverse (input) mode.

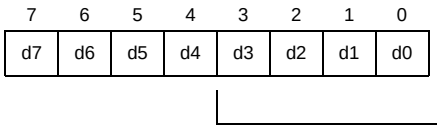


Data read from parallel output interface (reverse channel mode).

R_PAR_DATA<7:0> (write):

Address:	0x40	**Initial Value:**	Undefined

This is the parallel port 1 output interface data register. When automatic strobe is enabled, a write to this register will automatically transfer one byte to the printer. This register is double-buffered, which means that one byte can be written in advance, before the printer is ready to receive it.

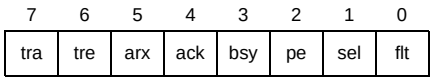


Data to printer.

R_PAR_STAT<7:0> (read):

Address: 0x41	**Initial Value:** N/A

This register contains the values of the different status signals of the parallel output interface, and some status bits.



$\overline{\text{Fault}}$ signal (active low). Also, in P1284/ BOISE nibble mode, data bits 0 then 4.

Select signal (active high). Also, in P1284/ BOISE nibble mode, data bits 1 then 5.

PE signal (active high). Also, in P1284/ BOISE nibble mode, data bits 2 then 6.

Busy signal (active high). Also, in P1284/ BOISE nibble mode, data bits 3 then 7.

$\overline{\text{Acknowledge}}$ signal (active low).

Acknowledge received status bit. This bit is set when an acknowledge is received, and reset when the par_data register is written. Normally used in manual strobe mode.

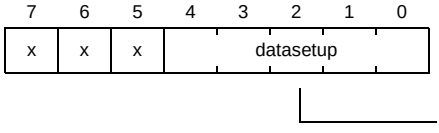
Transmitter empty status bit. When this bit is set, one byte can be written to par_data.

Transmitter active status bit. Indicates that the transmitter is busy sending data to the printer, or waiting for an acknowledge. Could be used to detect that the printer communication is hung.

R_PAR_DSETUP<4:0> (write):

Address: 0x41	**Initial Value:**	Undefined
Recommended setting: (Giving a data setup time of 1.1µs when the divide by five mode and a 32 MHz system clock is used.		0x06

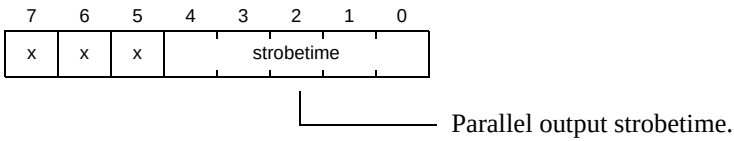
This register controls the time the parallel output data buffer is enabled before the strobe signal is activated. The basis for the counter is the Peripheral Clock controlled by the timing_mode register. The data setup time is approximately (par_dsetup + 1) * peripheral_clock_interval. With a system frequency of 32 MHz and the divide by five mode, peripheral_clock_interval is 1 / 32E+6 * 5 = 156.25 ns.



R_PAR_STROBE<4:0> (write):

Address: 0x42	**Initial Value:**	Undefined
Recommended setting:		0x06

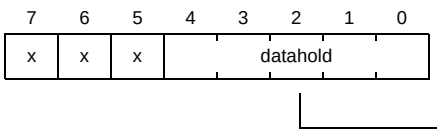
This register controls the length of the strobe signal. See the description of par_dsetup for further information.



R_PAR_DHOLD<4:0> (write):

Address: 0x43	**Initial Value:**	Undefined
Recommended setting:		0x06

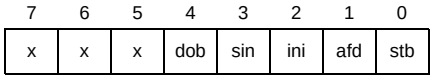
This register controls the time the parallel output data buffer is enabled after the strobe signal is deactivated. See the description of par_dsetup for further information.



R_PAR_CTRL<4:0> (write):

Address:	0x44	**Initial Value:**	0x04
		Recommended setting:	0x00

This register controls the different control signals in the parallel output interface.



Strobe signal. Only active when in manual strobe mode or when in input mode (see below). Note that the polarity of this bit is inverted when inverted strobe mode is selected. Normally, it is active high.

AutoFeed signal (active high).

Init signal (active high).

SelectIn signal (active high).

Data output buffer control (manual strobe mode) or data output buffer hold (automatic strobe mode).

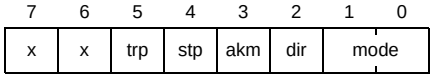
Manual strobe or input mode:
0: Data output buffer tristate.
1: Data output buffer enabled.

Automatic strobe mode:
0: Data on output port goes tristate after the data hold time has elapsed.
1: Data on output port is held until an acknowledge has been received.

R_PAR_CONFIG<5:0> (write):

Address:	0x45	**Initial Value:**	0x00
		Recommended setting:	0x01

This register controls the configuration of the parallel output interface.



Interface mode.
00: Parallel port disabled.
01: Centronics protocol.
10: IBM Fastbytes protocol.
11: Manual strobe mode.

Direction control.
0: Normal operation (output).
1: Input mode.

Acknowledge mode.
0: Acknowledge used.
1: Disregard acknowledge, only use busy, PE and select.

Strobe polarity.
0: Strobe active high.
1: Strobe active low. Note that this mode inverts the meaning of the manual strobe control bit in par_ctrl.

Parallel input hardware transparency mode.
0: Normal operation, CPU and/or DMA can access the parallel output port.
1: Data on parallel input port will be sent to parallel output port.

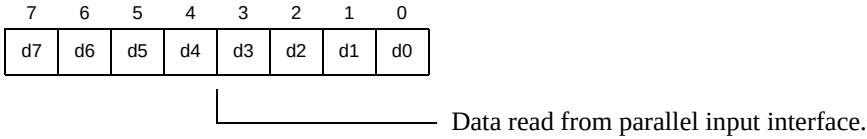
Note: It is possible to drive a Fastbytes printer in normal Centronics mode, but for maximum speed the Fastbytes mode should be used.

7.10 PARALLEL PORT2 REGISTERS

R_PCH_DATA<7:0> (read):

Address:	0x48	**Initial Value:**	N/A

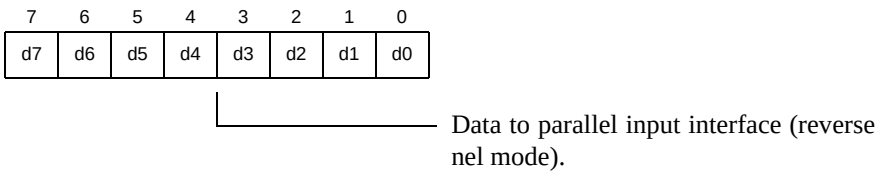
This register contains data read from the parallel input interface.



R_PCH_BACKCH<7:0> (write):

Address:	0x48	**Initial Value:**	Undefined

This is the data output register for the parallel input interface when used in reverse (output) mode.



R_PAR2_DATA<7:0> (write):

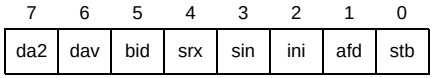
Address:	0x48	**Initial Value:**	Undefined

When parallel port 2 is used as an output port (by setting bit 5 in pch_config/ par2_config), this register is similar to R_PAR_DATA, but for parallel port 2.

R_PCH_STAT<7:0> (read):

Address:	0x49	**Initial Value:**	N/A

When parallel port 2 is used as an input port (by setting bit 5 in pch_config/par2_config to 0), this register contains the values of the different control signals of the parallel input interface, and some status bits.



$\overline{\text{Strobe}}$ signal (active low).

$\overline{\text{AutoFeed}}$ signal (active low).

$\overline{\text{Init}}$ signal (active low).

$\overline{\text{SelectIn}}$_signal (active low).

Strobe received status bit.

Bidirectional status bit. When this bit is 1, the host is trying to negotiate bidirectional mode.

Data available status bit. When this bit is 1, input data is available in pch_data. This bit is cleared when the pch_data register is read.

Data available #2. Indicates that there is incoming data on the parallel input interface. This bit is cleared when the pch_data register is read.

R_PAR2_STAT<7:0> (read):

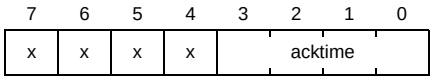
Address:	0x49	**Initial Value:**	N/A

When parallel port 2 is used as an output port (by setting bit 5 in pch_config/ par2_config to 1), this register is similar to R_PAR_STAT, but for parallel port 2.

R_PCH_ACKTIME<3:0> (write):

Address:	0x49	**Initial Value:**	0x0F
Recommended setting: (Giving an acknowledge length of 2 us with a crystal frequency of 32 MHz and Centronics mode.)			0x07
or: Giving an acknowledge length of 500 ns with a crystal frequency of 32 MHz and Fastbytes mode.)			0x03

When parallel port 2 is used as an input port (by setting bit 5 in pch_config/par2_config to 0), this register controls the length of the acknowledge signal. The length of the acknowledge signal in Centronics mode is (pch_acktime + 1) * 8 * fast_clock_interval. In Fastbytes mode, the length of the acknowledge signal is at least (pch_acktime + 1) * 4 * fast_clock_interval.



Parallel input acknowledge length.

R_PAR2_STROBE<4:0> (write):

Address:	0x49	**Initial Value:**	Undefined
Recommended setting:			0x06

When parallel port 2 is used as an output port (by setting bit 5 in pch_config/ par2_config to 1), this register is similar to R_PAR_STROBE, but for parallel port 2.

R_PAR2_DHOLD<4:0> (write):

Address:	0x4A	**Initial Value:**	Undefined
Recommended setting:			0x06

When parallel port 2 is used as an output port (by setting bit 5 in pch_config/ par2_config to 1), this register is similar to R_PAR_DHOLD, but for parallel port 2.

R_PAR2_BACKCH<7:0> (read):

Address:	0x4A	**Initial Value:**	N/A

When parallel port 2 is used as an output port (by setting bit 5 in pch_config/ par2_config to 1), this register is similar to R_PAR_BACKCH, but for parallel port 2.

R_PAR2_DSETUP<4:0> (write):

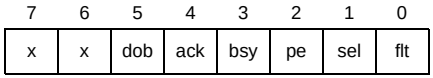
Address:	0x4B	**Initial Value:**	Undefined
		Recommended setting:	0x06

When parallel port 2 is used as an output port (by setting bit 5 in pch_config/ par2_config to 1), this register is similar to R_PAR_DSETUP, but for parallel port 2.

R_PCH_CTRL<5:0> (write):

Address:	0x4C	**Initial Value:**	0x0C
Recommended setting input port:			0x0C
Output port:			0x00

This register controls the different output signals in the parallel input interface.



Fault signal (active high). Also, in P1284/ BOISE nibble mode, data bits 0 then 4.

$\overline{\text{Select}}$ signal (active low). Also, in P1284/ BOISE nibble mode, data bits 1 then 5.

$\overline{\text{PE}}$ signal (active low). Also, in P1284/BOISE nibble mode, data bits 2 then 6.

$\overline{\text{Busy}}$ signal (active low). Also, in P1284/ BOISE nibble mode, data bits 3 then 7.

Acknowledge signal (active high). Only active when in manual acknowledge mode or when in output mode (see below).

Data output buffer tristate control. Only active when in manual acknowledge mode or when in output mode.
0: Data output buffer tristate.
1: Data output buffer enabled.

Note: The output signals have inverted polarity (compared to the Centronics specifications), and thus need inverters outside ETRAX.

R_PAR2_CTRL<4:0> (write):

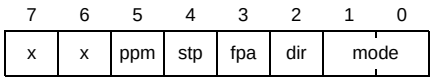
Address:	0x4C	**Initial Value:**	0x0C
Recommended setting:			0x00

When parallel port 2 is used as an output port (by setting bit 5 in pch_config/ par2_config to 1), this register is similar to R_PAR_CTRL, but for parallel port 2.

R_PCH_CONFIG<5:0>/ PAR2_CONFIG<5:0> (write):

Address:	0x4D	**Initial Value:**	0x00
Recommended setting input port:			0x09
Output port 2:			0x21

This register controls the configuration of the parallel input interface and the second parallel output interface.



Interface mode (both input and output port).
00: Parallel port disabled.
01: Centronics protocol.
10: IBM Fastbytes protocol.
11: Input port: Manual acknowledge mode.

Output port: Manual strobe mode.

Direction control of data bits 0-7 (both input and output port).
0: Normal operation.
1: Reversed (back channel);
parallel input port: output,
parallel output port: input.

Direction of $\overline{\text{FAULT2}}/\overline{\text{PCH\_PE}}$ pin (input port) or acknowledge mode (output port).
Input port:
0: Input pin.
1: Output pin (normal mode when not using parallel output port 2).
Output port:
0: Acknowledge used.
1: Disregard acknowledge, only use busy, PE and select.

Strobe polarity (output port).
0: Strobe active high.
1: Strobe active low. Note that this mode inverts the meaning of the manual strobe control bit in par2_ctrl.

Parallel port mode.
0: Parallel input port.
1: Parallel output port.

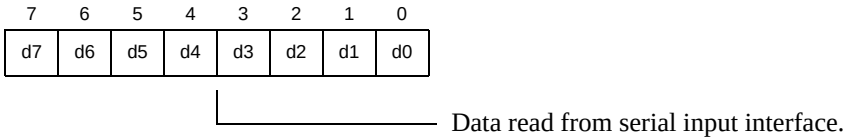
Note: In order to receive signals from a host using the Fastbytes protocol, the Fastbytes mode has to be selected.

7.11 SERIAL PORT REGISTERS

R_SER1_DIN<7:0> (read):

Address:	0x50	**Initial Value:**	N/A

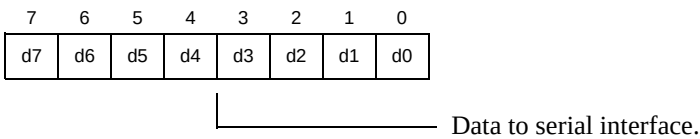
This register contains data read from the serial input interface.



R_SER_DOUT<7:0> (write):

Address:	0x50	**Initial Value:**	Undefined

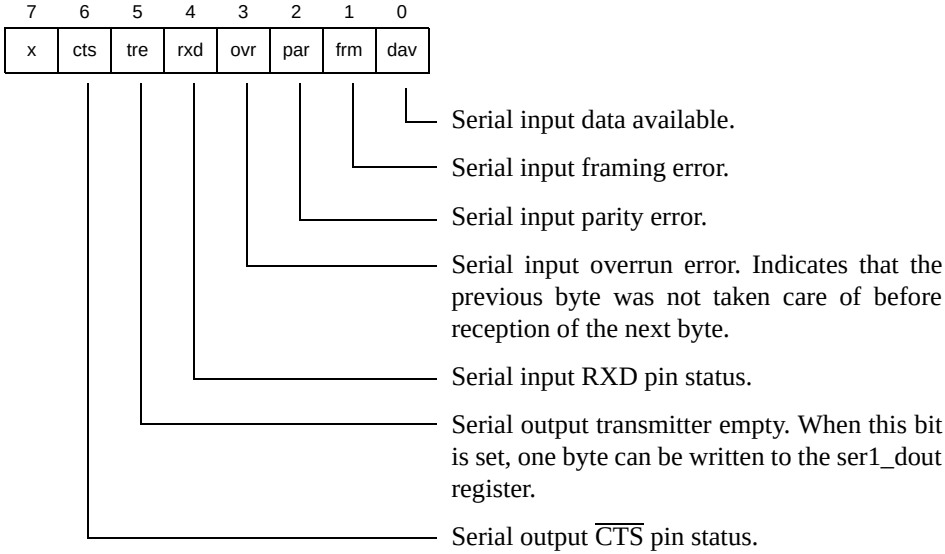
This is the serial output interface data register (double-buffered).



R_SER1_STAT<6:0> (read):

Address:	0x51	Initial Value:	N/A

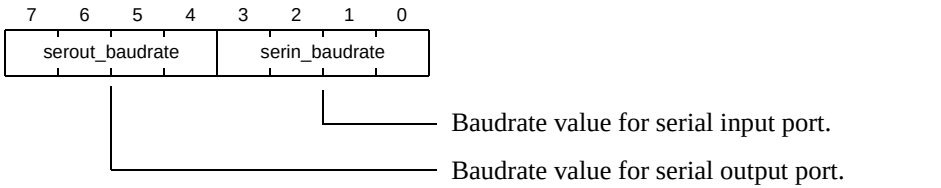
When parallel port 2 is used as an output port (by setting bit 5 in pch_config/ par2_config), this register contains the values of the different control signals of the parallel input interface, and some status bits.



R_SER1_BAUD<6:0> (write):

Address:	0x51	Initial Value:	Undefined

This register controls the baudrates of the serial input and output ports, respectively.



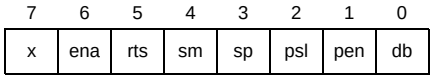
Value	Baudrate	Value	Baudrate	Value	Baudrate	Value	Baudrate	Value	Baudrate
0	300	2	1200	4	4800	6	19200	8	57600
1	600	3	2400	5	9600	7	38400	9	115200

Note: Higher values than 9 give unpredictable results.

R_SER1_IN_CFG<6:0> (write):

Address: 0x52	**Initial Value:** 0x00
Recommended setting: (8 data bits, no parity, $\overline{\text{RTS}}$ low)	0x40

This register controls the configuration of the serial input interface.



Number of data bits.
0: 8 data bits.
1: 7 data bits.

Parity enable.
0: Parity disabled.
1: Parity enabled.

Parity select.
0: Even parity.
1: Odd parity.

Stick parity mode select.
0: Normal parity (even or odd).
1: Stick parity enabled. Parity is checked as a logic 0 or 1, depending on bit 2 in this register.

Sampling mode.
0: One sample in middle of data bit.
1: Majority of three samples in middle of data bit.

RTS_ control.
0: RTS_ pin low.
1: RTS_ pin high.

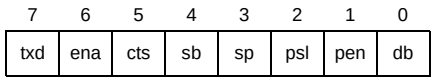
Enable serial input interface.
0: Serial input port disabled.
1: Serial input port enabled.

Note: R_SER_CONNECT must be set up correctly for the serial interface to work.

R_SER1_OUT_CFG<7:0> (write):

Address: 0x53	**Initial Value:**	0x00
Recommended setting: (8 data bits, no parity, 1 stop bit, autoCTS		0x60

This register controls the configuration of the serial output interface.



Number of data bits.
0: 8 data bits.
1: 7 data bits.

Parity enable.
0: Parity disabled.
1: Parity enabled.

Parity select.
0: Even parity.
1: Odd parity.

Stick parity mode select.
0: Normal parity (even or odd).
1: Stick parity enabled. Parity is transmitted as a logic 0 or 1, depending on bit 2 in this register.

Number of stop bits.
0: 1 stop bit.
1: 2 stop bits.

Enable automatic CTS_ handling.
0: CTS_ disregarded.
1: CTS_ active; CTS_ high stops transmission after the ongoing byte.

Enable serial output interface.
0: Serial output port disabled.
1: Serial output port enabled.

TXD pin manual control.
0: TXD pin low.
1: TXD pin high.

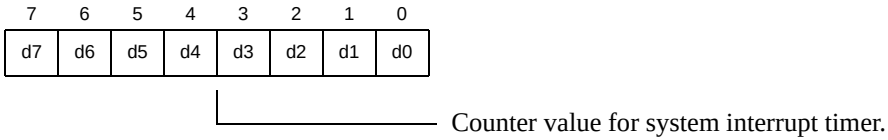
Note: TXD pin manual control is only active when the serial output interface is disabled (by setting bit 6 in this register to "0"). Also note that R_SER1_CONNECT must be set up correctly for the serial interface to work.

7.12 TIMER REGISTERS

R_TIMER_DATA<7:0> (write):

Address:	0x58	**Initial Value:**	Undefined

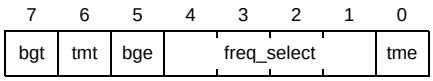
This is the counter value for the system interrupt timer. The timer will generate an interrupt each time the decrementing counter reaches 1, and on the next counter cycle start with the value in this register. Note that the interrupt must be acknowledged by a write to timer_inta. The cycle length is determined by the frequency selected in the timer_mode register.



R_TIMER_MODE<7:0> (write):

Address:	0x59	**Initial Value:**	0x00

This register contains mode bits and basic frequency select for the timer, as well as a mode bit for the baudrate generator.



Enable timer.
0: Timer disabled.
1: Timer enabled.

Timer basic frequency select. These bits control which frequency is used as the basis for the counter in the timer. The values 0-10 select frequencies of 300 up to 921600 Hz.

Value	Frequency	Value	Frequency
0	921600	6:	9600
1	300	7	19200
2	600	8	38400
3	1200	9	57600
4	2400	10	115200
5	4800	11-15	Do not use

Enable baudrate generator.
0: Baudrate generator disabled.
1: Baudrate generator enabled.

Timer test mode.
0: Normal mode.
1: Special test mode.

Baudrate generator test mode.
0: Normal mode.
1: Special test mode.

Note 1: In order for the timer and baudrate generator to operate properly, the timer_data and ser1_baud registers must be loaded with the correct values and then at least 45 fast clock cycles must elapse before the timer and baudrate generator are enabled (by setting the enable bits in this register).

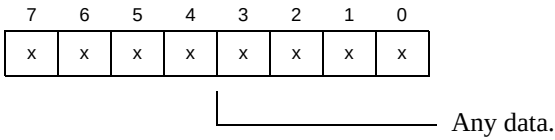
Note 2: For the timer to operate, the baudrate generator must be enabled, by setting bit 5 in this register.

Note 3: The test modes for timer and baudrate generator should normally not be used, and are therefore not described here.

R_TIMER_INTA<7:0> (write):

Address:	0x5A	**Initial Value:**	N/A

This is the interrupt acknowledge port for the system timer. A write to this port will reset the timer interrupt. Data written is don't care.



7.13 ETHERNET AND TOKEN RING REGISTERS

Station address registers

Token-ring and ethernet addresses are described as a bit sequence with the highest numbered bit (msb) corresponding to the first transmitted bit. The msb is the leftmost bit when written in binary representation. The '.' is used as a concatenation operator for bit fields. The notation xxx<4:2> is equivalent to xxx<4> . xxx<3> . xxx<2> and xxx<2:3> is equivalent to xxx<2> . xxx<3>. The destination address field in a frame will in the following be refered to as addr<47:0>.

The address registers behaves differently in token-ring and in ethernet mode. Some of the bits in these registers also function as mode bits for the address interpretation.

A hint is to first find out the bit order in which the address will appear on the network cable and then map that order to the address registers.

R_MA0 - R_MA5 (write):

Address:	0x80-0x85	**Initial Value:**	Undefined

These registers are described in section 4

R_GA0 - R_GA7 (write):

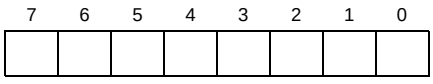
Address:	0x88-0x8F	**Initial Value:**	Undefined

These registers are described in section 4.

R_TR_MODE1<7:0> (write):

Address: 0x90	**Initial Value:** Undefined
Recommended setting:	–

Configuration register for network interface.



act_mon: ***Note 1***
1: active monitor mode.

opr_act: ***Note 1***
1: operational active.

promiscuous: ***Note 2***
1: Promiscuous mode, i.e. frames to all addresses are accepted.

fill_en, in token-ring mode:
1: Repeat mode is disabled and fill is continuously sent. ***Note 1***
backoff_test, in ethernet mode:
1: Back off timer will count at high speed. For factory testing only.

etr, in token-ring mode: ***Note 2***
1: Early token release is enabled.
loop-back, in ethernet mode:
0: Ethernet receiver is enabled during transmission. Used in loop back mode.
1: Receiver is disabled during transmission.

ethernet: ***Note 2***
1: Network interface in ethernet mode.
0: Network interface in token-ring mode.

send_wo_token: ***Note 1***
1: Frames in transmit buffer will be sent immediately without awaiting token.

send_token: ***Note 1***
0->1 Positive flank on this bit will send a token.

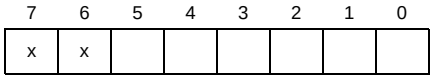
Note 1: This bit must be high/low for at least 8 bit times.

Note 2: This bit must only change value when etr_reset_ (R_TR_MODE2<2>) is low!

R_TR_MODE2<4:0> (write):

Address:	0x91	**Initial Value:**	0x00
		Recommended setting:	–

Config register for network interface.



rd_intr_status:
Positive flank on this bit will load current interrupts into R_ETR_STATUS and clear those interrupts. R_ETR_STATUS isn't valid until 16 net bit times after the flank and will continue to be valid until rd_intr_status is brought low. New interrupts will be delayed until rd_intr_status is brought low.

elastic_buffer: ***Note 1***
1: Insert elastic buffer.
0: Remove elastic buffer.

etr_reset_
0: Hold network interface in reset.
1: Starts network interface.

insert:
1: Insert into ring. Generate a 2400 Hz square wave on NSRT_.

enable_adr: ***Note 2***
1: Address recognition enabled.
0: Address recognition disabled.

inter frame gap:
0: Inter frame gap disabled.
1: Inter frame gap enabled.

Note 1: This bit must be high/low for at least 8 bit times.

Note 2: When disabled it's allowed to change the R_MAx and R_GAx registers except for the mode bits in these registers. When address recognition is disabled only broadcast frames are copied. No frames should be sent using the normal send with token procedure (send_wo_token is ok) since MA_FLAG wont work.

R_TVX<5:0> (write):

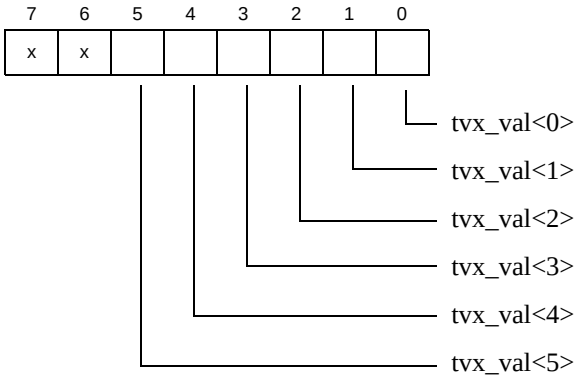
Address:	0x92	**Initial Value:**	Undefined
		Recommended setting:	0x18

When the timer expires an interrupt is generated, see R_ETR_STATUS.

Time out value: (tvx_val<5:0> / 2400) sec.

IEEE specifies TVX to 10.0 ms, i.e. the tvx_value should be 0x18.

This register must only change value when $\overline{\text{etr\_reset}}$ (R_TR_MODE2<2>) is low.



Note: The baudrate generator must be enabled before the TVX and TRR timers are started.

R_TRR<5:0> (write):

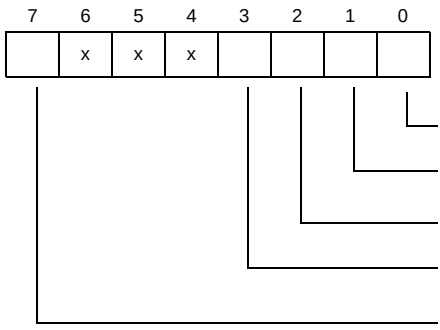
Address:	0x93	**Initial Value:**	Undefined
		Recommended setting:	0x0A

When the timer expires an interrupt is generated, see R_ETR_STATUS.

Time out value: (trr_val<3:0> / 2400) sec.

IEEE specifies TRR to 4.0 ms, i.e. the tvx_value should be 0x0A.

Trr_val<3:0> must only change value when etr_reset_ (R_TR_MODE2<2>) is low. To restart the timer by setting bit 7 is, however, allowed.



trr_val<0>

trr_val<1>

trr_val<2>

trr_val<3>

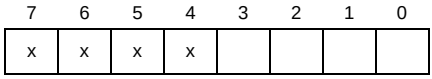
restart_trr:
1: TRR is reloaded with the start value
0: TRR counting is enabled.
This bit must be held high/low for at least 16 bit times.

Note: The baudrate generator must be enabled before the TVX and TRR timers are started.

R_ANALOG<3:0> (write):

Address:	0x94	**Initial Value:**	Undefined
		Recommended:	–

Control signals to the analog interface circuits.



spsw/mode: Directly connected to the spsw/mode pin.

spsw: (in token-ring mode)
0: 16 Mbit/s
1: 4 Mbit/s
Speed switch 4/16 MHz for the analog circuit. Should be connected to SPSW pin on TMS38054.

mode: (in ethernet mode)
Mode select (SEL) signal to the analog circuit, DP8391A.

fraq:

Frequency/phase acquisition mode in token-ring mode. Directly connected to FRAQ pin.
1: frequency mode
0: phase mode

In ethernet mode the FRAQ pin is used as input for COL (collision detected) from DP8391A, i.e. it can't be used as a general output port.

clr_tok_recvd: ***Note***
1: Token_recvd (R_REC_STATUS<4>) is cleared.

fast_tick05ms:
1: TVX and TRR timers will use rd_wr as timer tick. For factory testing only.

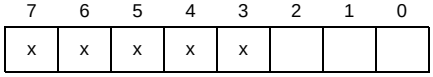
Note: This bit must be high/low for at least 8 bit times.

7.14 PACKET RECEIVE AND TRANSMIT REGISTERS

R_REC_MODE<2:0> (write):

Address:	0x98	**Initial Value:**	0x01
		Recommended:	–

Receiver Ring Buffer mode register.



reset_rec:
0: Enable receiver.
1: Reset and disable receiver.

clr_buf_full:
0: No effect.
1: Clears the buffer full status bit (R_REC_STATUS<0>) and the Buffer full interrupt (R_BUF_STATUS<2>) ***Note***

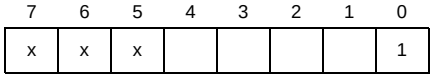
clr_pkt_recvd:
0: No effect.
1: Clears the packet received interrupt (R_BUF_STATUS<0>.) ***Note***

Note: These bits only have effect directly when the register is written to, i. e. they don't have to be set to 0 again to re-enable the interrupts

R_REC_STATUS<4:1> (read):

Address:	0x98	**Initial Value:**	N/A

Receiver Ring Buffer Status Register.



1

buffer_full:
1:Buffer full.

sync: (in token-ring mode)
0: Enough transition density on receive data for TMS38054 to lock.
1: Too low transition density on receive data for TMS38054 to lock.

This bit must be zero for receive data to be valid. Directly connected to $\overline{\text{REDY}}$ pin.

cs: (in ethernet mode)
CRS (carrier sense) from DP8391A.

fraq: (in token-ring mode)
To read FRAQ pin.

cd: (in ethernet mode)
COL (collision detect) from DP8391A.

token_recvd: (in token-ring mode)

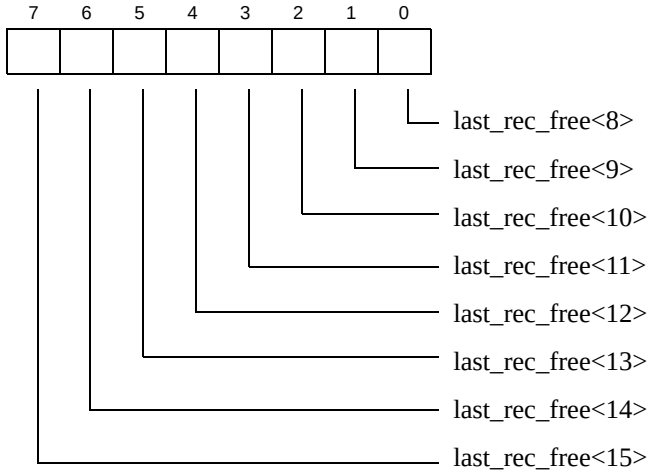
A correct token has been seen since last time this bit was cleared. To clear this bit use clr_tok_recvd (R_ANALOG<2>).

R_REC_END<7:0> (write):

Address:	0x99	**Initial Value:**	Undefined
		Recommended setting:	–

Points to the end of free space in the receive ring buffer.

Bits <15:8> of address to last free 256 byte block.



R_REC_POS<7:0> (write):

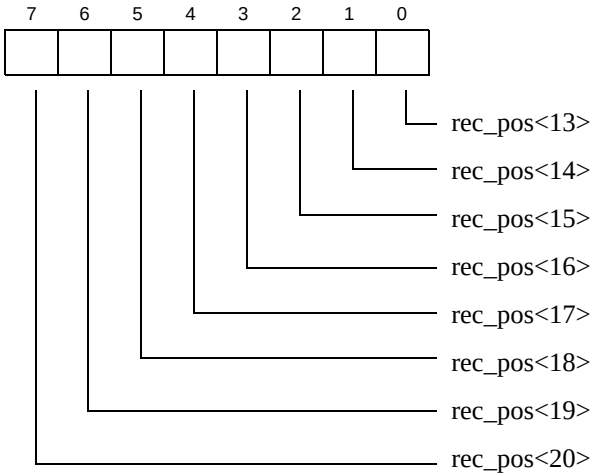
Address:	0x9A	**Initial Value:**	Undefined
		Recommended setting:	–

Placement of receive ring buffer.

Bits <20:13> of start of receiver ring buffer.

Bit <20:13> are used for 8Kbyte buffers.
Bit <20:14> are used for 16Kbyte buffers.
Bit <20:15> are used for 32Kbyte buffers.
Bit <20:16> are used for 64Kbyte buffers.

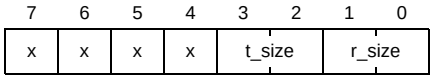
All other bits must be set to 0.



R_RT_SIZE<3:0> (write):

Address:	0x9B	**Initial Value:**	Undefined
		Recommended setting:	–

Sets the sizes of the receive and transmitt ring buffers.



receive buffer size:

R_RT_SIZE<1:0>	**size(KB)**
00	8
01	16
10	32
11	64

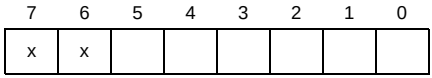
transmit buffer size:

R_RT_SIZE<3:2>	**size(KB)**
00	2
01	4
10	8
11	64

R_TR_CMD<5:0> (write):

Address: 0x9C	**Initial Value:** 0x01
Recommended setting:	–

Transmit command register.



reset:
0: Enable.
1: Reset. Stop transmission immediately.

retransmit: (in ethernet mode)
0: Make no transmission retries.
1: Make 15 transmission retries.
Should always be zero in token ring mode.

new_token:
0: Allow several packets to be sent on the same token.
1: Always await a new token for each packet to be transmitted.

pkt_intr_en:
0: No interrupt after transmitted packet.
1: Interrupt after all packets in the buffer have been transmitted.

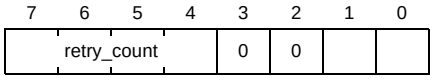
transmit:
0: Stop transmission after current packet.
1: Start transmission of packets in the transmit ring buffer.

cancel:
0: Normal operation.
1: Cancel pending transmit buffers.
This can be used to give higher priority packets precedence in token-ring.

R_TR_STATUS<7:0> (read):

Address:	0x9C	**Initial Value:**	N/A

Buffer handler transmit status register.



transmitting:
0: Stopped.
1: Transmitting

abort:
0: Not aborted.
1: Transmission aborted due to excessive errors.

0

0

retry_count:

The number of retries done so far for current packet.

R_TR_START<7:0> (read/write):

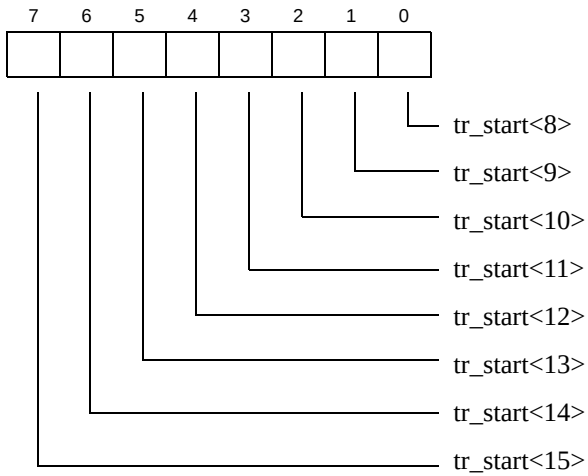
Address:	0x9D	**Initial Value:**	Undefined
		Recommended setting:	–

Bits <15:8> of the start address of the current package to be transmitted.

Note the overlap between R_TR_START and R_TR_POS in bits 15, 14 and 13.

Bit <12:8> is used for 2,4 and 8 Kbyte buffers. Bit <15:13> must match the value in R_TR_POS.

Bit <15:8> is used for 64 Kbyte buffers.



R_TR_POS<7:0> (write):

Address:	0x9E	**Initial Value:**	Undefined
	Recommended setting:		–

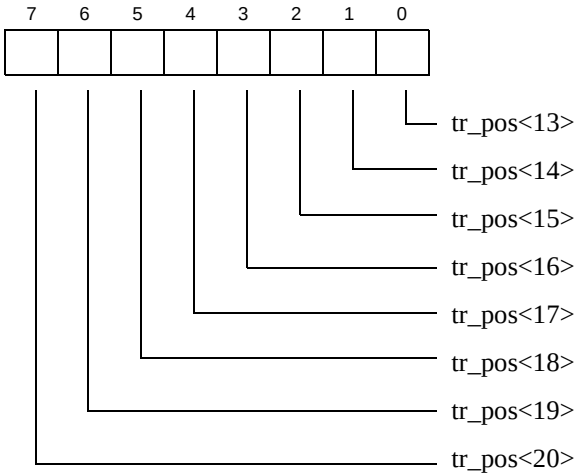
Placement of transmit ring buffer.

Bits <20:13> of the start address of the transmit ring buffer.

Note the overlap between R_TR_START and R_TR_POS in bits 15, 14 and 13.

Bit <20:13> is used for 2,4 and 8 Kbyte buffers.

Bit <20:16> is used for 64 Kbyte buffers. Bit <15:13> must be set to 0.



APPENDIX A: MGMON

A.1 GENERAL DESCRIPTION

MGMON is a small machine monitor for microprocessors, which includes all the common functions of a monitor, like printing and editing memory, disassembling and running programs. It was orginally written as a 6809 monitor, but have been ported to ETRAX and UNIX systems. For historical reasons this version also includes an optional 6809 simulation mode.

It is designed for easy porting between different systems, and implements all basic printing and reading in the monitor code, which makes it almost independable of the C-libraries functions. It is also intended to be easy to use for persons with knowledge of microprocessor system.

On an ETRAX system, the monitor is used via serial port 1, at 9600 bps, 8N1. This is enough for normal use, since you don't download any software through the serial port. All large data transfers are done via the parallell port. Changing the terminal speed requires recompilation of MGMON.

A description of MGMON's commands, and some short notes about the sourcecode is given below. Also try "help" and "info" inside the monitor for online help.

A.2 STANDARD COMMANDS

The command syntax in MGMON is very simple. All commands are listed with 'info', which gives a list of all available commands with syntax and a short description of the command. All commands can be abbreviated with a blank space or a dot. When a dot is used to terminate a word, the first match in the info-list order is choosen, otherwise there must be enough letters to make the command unique. No commands perform syntax checking, and all skipped parameters will default to zero, so it is the users responsibility to not forget any parameterts. Note that all end addresses should be specified as first address not to be included., i.e. add one to all stop addresses. If a memory offset is specified with the base command, it will be added to all addresses specified.

```
list
```

```
list [start] [stop] [width]
```

Lists the memory from <start> to <stop> in hex and ascii format formatted in <width> columns. If no stop address is specified, list will work interactively with eight columns per line. Space lists a new line, and CR stops list.

```
help
```

Gives a short summary of this document.

```
info
```

Lists all commands, with parameters and short description.

```
base
base [address]
```

Sets a memory offset, which will be added to all addresses specified. This is useful for large address, and saves a lot of typing. Without parameter base will show current memory offset.

```
dis
dis [start] [stop]
```

Disassemble cris-instructions from <start> to <stop>. If stop isn't specified dis works interactively, disassembling line by line. CR stops the disassembling.

```
program
program [address]
```

Program is used to write data into memory. Program prints the current memory address and its contents, and then waits for data.

Data can be writter either as hex bytes, seperated with space, or as an ascii string prepended with a quote (") .

The address counter is incrented/decrented with +/-.

Use a single dot to exit program mode.

```
write
write [address]
```

Works like program, but the memory contents at the current address is not printed.

```
call
call address [repeats]
```

Executes the function at specified address. The number of calls can be specified with <repeats> (default is one).

```
move
move start stop destination
```

Moves a memory block from start to destination. This routine will not handle destinations within the source region. Note that the stop address is not copied and that no parameters is checked.

```
fill
fill start stop data
```

Fills memory from start to stop-1 with the byte data.

```
compare
```

```
compare start1 stop start2
```

Compares the memory block at start1 with start2. If a mismatch is found, the address and data will be shown. Space will continue the search, CR stops it.

```
find
```

```
find pattern start
```

Search for data in memory for specifies data. <pattern> is a pointer to the data to search for, with the first byte specifying how many bytes the pattern contains, excluding the first byte. The start location is specified with <start>, and memory is searched 0x1000 bytes forward.

```
modify
```

```
modify address data
```

Writes byte <data> at <address>.

```
check
```

```
check [start] [stop]
```

Calculates a 16-bit CRC checksum. Note that stop is first byte not to include.

```
load
```

```
load [address] [filename] (Unix only parameters)
```

Loads a program into memory. If the system is an ETRAX, data will be loaded from the parallell port into DRAM at 0x60000000. If 128 KB is loaded the program will auto-start, otherwise load is stopped by pressing a key.

On a UNIX system this will load specified file to <address>.

```
save
```

```
save [start] [stop] [filename]
```

Only works on UNIX systems. Saves specified memory block to a file.

```
display
```

Shows the CPU registers in ETRAX. This will of course only work on an ETRAX system. The shown registers are:

```
DMA channel 0 Transfer Pointer (DTP0)
```

```
DMA channel 0 Transfer Counteri (DCR0)
```

```
Interrupt return pointer (IRP)
```

```
Subrouine return pointer (SRP)
```

```
Program Counter (PC)
```

```
Stack Pointer (SP)
```

A.3 6809 COMMANDS

The following commands are only available when MGMON is compiled to support 6809.

`registers`

Shows the 6809 registers from the simulator

`set`

`set reg value`

Sets a 6809 simulator register to <value>. Note that reg should only be one letter, i.e. 'set p 1000' will set the pc register to 0x1000.

`run`

`run address`

Run will execute the program at <address> in the 6809 simulator.

`ass09`

`ass09 address`

Assemble 6809 program. Use +/- to increment/decrement address.

`dis09`

`dis09 address`

Disassembles 6809 programs. Use space to see next line, CR to stop.

`exit`

Exit leaves the monitor and works only under UNIX. Leaving the monitor this way instead of sending a break is important if your terminal is sensitive to the stty changes MGMON performs.

A.4 SOURCE CODE

The source code for MGMON is split into several files. The files have a short explantion below, describing there contents, and what they do. Note that MGMON was orginally written in assembler 1981, and later ported into C. That is why many functions available in modern C-compilers are implemented in the monitor. This makes it rather simple to port MGMON to systems where no C-libaries are available.

Most modules here are independent of the implementation of the other modules, except for the 6809 part and some functions in mgmon.c, which heavily depends on the io.c file.

A.4.1 The makefiles

There are four different makefiles to build MGMON. They can all be called from the common makefile, but typing "make <target> where target is etr6809, etrax, unix6809

or unix, for ETRAX and UNIX versions with or without 6809 simulation. It can be required to remove the object files between compilation, when the 6809 switch is changed.

A.4.2 mgmon.c

This is the main monitor file. It contains all system-independent functions, like command reading and writing in the memory in a safe way, command line parsing, and most of the commands found in mgmon. The commands not found here is the 6809 simulation commands, and commands which are completly different on varoius platforms, like loading, and system setup. They are instead found in unix_io.c, etrax_io.c and in the 6809 files.

A.4.3 io.c

This file contains basic io-routines for printing different kinds of variables, and buffering input. The routines were written before printf was included, and could today be replaced with printf calls instead. This is not the case yet though.

A.4.4 etrax_io.c

Here is all low-level hardware routines for ETRAX. "mginit", which sets up baud rates, clock speed and RAM access times is found here. The module also contains routines for reading and writing to the serial port, and loading program code through the parallell port.

A.4.5 unix_io.c

This is a replacement for etrax_io.c when running on an UNIX system. It is mostly "dummy" functions, since almost everything is set up already.

A.4.6 crt0.c

A very simple bootstrap code for ETRAX. This must be the first module to the linker, since the startup-code must begin at address 0x00000002 in ETRAX. This file also shows how to use interrupts in a simple way. The interrupt code is currently unused in MGMON, since all events are polled.

A.4.7 6809.c dis09.c asm09.c sim09.c

These file includes all software for the 6809 functions in MGMON. Note that this is an unsupported feature in MGMON. 6809.c is the main file, containging all shared variables, and all monitor commands for the 6809 is found here.