

PMC-FIFO

2 * 16bit FIFO (4k, 16k or compatible)

8bit TTL-Input with IRQ

8bit TTL-Input Static

8bit TTL-Output

Software Manual

NOTE

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

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

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

esd electronic system design gmbh

Vahrenwalder Str. 205
30165 Hannover
Germany

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

USA / Canada

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

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

Software described	Revision/date
PMC-FIFO C-Interface	05/05/99

Changes in the software and/or documentation

Changes in this manual as compared with previous version	Changes in the software	Changes in the documentation
first english issue	-	-

Contents	Page
1. Programming Interface	3
1.1 Initialisation	3
pmcfifoOpen()	3
pmcfifoClose()	3
1.2 Reading the FIFOs	4
pmcfifoFIFOConfig()	4
pmcfifoFIFOClear()	5
pmcfifoFIFOReset()	5
pmcfifoFIFORead()	5
pmcfifoFIFOStatus()	6
1.3 Interrupt Handling	7
pmcfifoIRQSignal()	7
pmcfifoIRQWait()	7
pmcfifoIRQRead()	8
pmcfifoTTLLRead()	9
pmcfifoTTLWrite()	9
 2. Returned Values	 10

1. Programming Interface

The following chapter will describe the C-programming interface of the PMC-FIFO. The values returned in case of errors will be described in chapter 2.

1.1 Initialisation

The services described below are used to initialise the PMC-FIFO hardware.

pmcfifoOpen()

Name: **pmcfifoOpen()** - Initialising the PMC-FIFO board

Call: `int pmcfifoOpen
(
)`

Description: This function enables the memory area of the PMC-FIFO and initialises the PCI-components on the host CPU and the PMC-FIFO board. It has to be called before any other routine for communication with the PMC-FIFO.

Return: 0 or an error code described in the appendix.

pmcfifoClose()

Name: **pmcfifoClose()** - Terminating the communication with the PMC-FIFO board.

Call: `int mcfifoClose
(
)`

Description: This function switches off all connected interrupts and blocks all further access to the PMC-FIFO.

Return: 0 or an error code described in the appendix.

1.2 Reading the FIFOs

The services described below configure and communicate with the FIFO-components on the PMC-FIFO.

pmcfifoFIFOConfig()

Name: **pmcfifoFIFOConfig()** - Configuring the FIFO-memory.

Call:

```
int    pmcfifoFIFOConfig
(
    int    i,           /* FIFO-number (1 or 2) */
    int    mode,       /* mode (0 ... 2) */
    int    depth,      /* FIFO depth (e.g. 4096) */
    void * buf         /* pointer to free buffer */
)
```

Description: This function configures the FIFO-buffer *i* as 8 (*mode=0*), 16 (*mode=1*) or 32 (*mode=2*) bit component. An interrupt service routine is connected which guarantees that the FIFOs are automatically read-out in status 'half full' or 'full'. The parameter *depth* has got the depth of the FIFO-components used. This value has to be specified, because the components can be exchanged. In order to save the FIFO values a pointer to a buffer *buf* large enough is specified. The first longword in buffer *buf* contains the write pointer, which shows the number of values which have been read out of the FIFO.

In 8-bit mode the 8 MSB are ignored and only the 8 LSB are saved.

In 32-bit mode the two FIFO-buffers (each 16 bit) are combined to a 32-bit component and saved as a 32-bit value. FIFO number 1 has got the 16 LSB and FIFO number two has got the 16 MSB. 1 has to be specified for FIFO number *i*. In 32-bit mode it is not permissible to configure FIFO number 2. In this configuration the FIFO control signals (WRITE*, RESET*) of the first FIFO are used as control signals for both FIFOs.

The FIFO-buffer is deleted additionally.

Return: 0 or an error code described in the appendix.

pmcfifoFIFOClear()

Name: **pmcfifoFIFOClear()** - Deleting a FIFO-buffer.

Call:

```
int    pmcfifoFIFOClear
      (
        int    i           /* FIFO-number (1 or 2) */
      )
```

Description: This function deletes one of the two FIFO-memory components of the PMC-FIFO.

Return: 0 or an error code described in the appendix.

pmcfifoFIFOReset()

Name: **pmcfifoFIFOReset()** - Resetting a FIFO-buffer.

Call:

```
int    pmcfifoFIFOReset
      (
        int    i,           /* FIFO-number (1 or 2) */
        int    mode        /* mode (0 ... 2) */
      )
```

Description: This function deletes one of the two FIFO-memory components of the PMC-FIFO and resets the FIFO again with the mode specified (see **pmcfifoFIFO-Config**).

Return: 0 or an error code described in the appendix.

pmcfifoFIFORead()

Name: **pmcfifoFIFORead()** - Complete read-out of a FIFO-buffer.

Call:

```
int    pmcfifoFIFORead
      (
        int    i,           /* FIFO-number (1 or 2) */
        int    signal,      /* signal code */
        int    proc_id     /* process ID */
      )
```

Description: This function reads out the FIFO-buffer *i* completely (until the FIFO-component signals 'empty'). Then the signal *signal* is transmitted to the process *proc_id* by calling *_os_send()*.

Return: 0 or an error code described in the appendix.

pmcfifoFIFOStatus()

Name: **pmcfifoFIFOStatus()** - Reading the status of the FIFO-buffer.

Call: `char pmcfifoFIFOStatus
(
)`

Description: This function reads the status of the two FIFO-buffers and shows their value as return parameter.

Return: 8 bit with the following meaning:

Bit 0: FIFO 1 empty	Bit 4: FIFO 2 empty
Bit 1: FIFO 1 half full	Bit 5: FIFO 2 half full
Bit 2: FIFO 1 full	Bit 6: FIFO 2 full
Bit 3: reserved	Bit 7: reserved

1.3 Interrupt Handling

The services described below are used to configure the inputs which are suitable for interrupts.

pmcfifoIRQSignal()

Name: **pmcfifoIRQSignal()** - Configuring the inputs suitable for interrupts (OS-9).

Call:

```
int    pmcfifoIRQSignal
(
    int    mode,          /* mode (0 or 1) */
    int    bit,          /* bit number (0 ... 7) */
    int    signal,       /* signal code */
    int    proc_id       /* process ID */
)
```

Description: This function links the interrupt input *bit* (0...7) with *signal*.

If 1 is specified as *mode*, a *_os_send(proc_id, signal)* is once executed, provided the corresponding interrupt is triggered. This means that a process waiting for this signal can be 'pushed' once via an interrupt.

If 2 is specified as *mode*, the link is not established once but permanently. This link, however, can be cancelled again by calling *mode* 0.

If 0 is specified as *mode*, the link between *bit* and *signal* is cancelled again. No process is executed by the specified interrupt *bit*.

Return: 0 or an error code described in the appendix.

pmcfifoIRQWait()

Name: **pmcfifoIRQWait()** - Waiting for an interrupt (VxWorks / RTOS-UH).

Call:

```
int    pmcfifoIRQWait
(
    int    bit          /* bit number (0 ... 7) */
)
```

Description: After this function has been called the calling process waits until an interrupt is generated at the specified interrupt input *bit* (0...7).

Return: 0 or an error code described in the appendix.

pmcfifoIRQRead()

Name: **pmcfifoIRQRead()** - Reading the 8 interrupt inputs.

Call: **int** **pmcfifoIRQRead**
 (
)

Description: This function reads the eight IRQ-inputs and gives their interrupt status as return parameters. A 1 corresponds to a previous ‘high-to-low’ level change.

Return: The 8 bits combined in a least significant byte in integer. Bit 0 corresponds to input number 0, etc.

1.4 Reading and Writing Data

The services described below are used to read the TTL-inputs or to write the TTL-outputs.

pmcfifoTTLRead()

Name: **pmcfifoTTLRead()** - Reading the 8 TTL-inputs.

Call:

```
int    pmcfifoTTLRead
      (
      )
```

Description: This function reads the 8 TTL-inputs and gives their value as return parameters. A 1 corresponds to a 'high' and a 0 to a 'low' at the input.

Return: The 8 bits combined in a least significant byte in integer. Bit 0 corresponds to input number 0, etc.

pmcfifoTTLWrite()

Name: **pmcfifoTTLWrite()** - Writing one of the 8 TTL-outputs.

Call:

```
int    pmcfifoTTLWrite
      (
      int    bit_no,      /* bit number (0 ... 7) */
      int    bit_count,  /* bit counter (1 ... 8) */
      int    value       /* value (0 or 1) */
      )
```

Description: This function configures one or more of the 8 TTL-outputs (from bit number *bit_no* *bit_count*-bits) to the specified value *value*. A 1 corresponds to a 'high' and a 0 to a 'low' at the output. The other outputs remain unchanged.

Return: 0 or an error code described in the appendix.

