The Embedded I/O Company

TEWS TECHNOLOGIES

TPMC821-SW-42

VxWorks Device Driver

INTERBUS Master G4 PMC

Version 1.4

User Manual

Issue 1.2

January 2004

TEWS TECHNOLOGIES GmbH
Am Bahnhof 7 25469 Halstenbek / Germany
Phone: +49-(0)4101-4058-0 Fax: +49-(0)4101-4058-19
e-mail: info@tews.com www.tews.com

TEWS TECHNOLOGIES LLC
1 E. Liberty Street, Sixth Floor Reno, Nevada 89504 / USA
Phone: +1 (775) 686 6077 Fax: +1 (775) 686 6024
e-mail: usasales@tews.com www.tews.com

TPMC821-SW-42

INTERBUS Master G4 PMC

VxWorks Device Driver

Issue	Description	Date
1.0	First Issue	January 2000
1.1	Support for Intel x86 based targets	June 2000
1.2	General Revision	January 2004

Table of Content

1 Introduction

The TPMC821-SW-42 VxWorks device driver allows the operation of the TPMC821 PMC conforming to the VxWorks system specification. This includes a device-independent basic I/O interface with *open*(), *read(), write()* and *ioctl()* functions.

Special I/O operation that do not fit to the standard I/O calls will be performed by calling the *ioctl()* function with a specific function code and an optional function dependent argument.

This driver invokes a mutual exclusion and binary semaphore mechanism to prevent simultaneous requests by multiple tasks from interfering with each other.

To prevent the application program for losing data, incoming messages will be stored in a message FIFO with a depth of 100 messages.

This device driver supports the following features:

- use all possible bus operation modes
    - asynchronous mode without consistency locking
    - asynchronous mode with consistency locking
    - bus synchronous mode
    - program synchronous mode
- use bit commands
- use mailbox commands
- read data
- write data
- control the host interrupt request
- reset hardware error

2 Installation

The software is delivered on a 3½" HD diskette.

Following files are located on the diskette:

tp821drv.c	TPMC821 Device Driver Source
tpmc821.h	TPMC821 Include File for driver and application
tp821def.h	TPMC821 Driver Include File
tp821exa.c	TPMC821 Example Application
tpmctime.h	Include file with delay macro
tpmc_pci.c	PCI dependent functions
tpmc_pci.h	PCI dependent include
tp821_pci.c	TPMC821 PCI MMU mapping for Intel x86 based targets
tpxxxhwdep.c	Collection of hardware dependent functions
tpxxxhwdep.h	Include for hardware dependent functions

For installation the files have to be copied to the desired target directory.

2.1 Install the driver to VxWorks system

To install the TPMC821 device driver to the VxWorks system following steps have to be done:

- Build the object code of the TPMC821 device driver
- Link or load the driver object file to the VxWorks system
- Call the *tp821Drv()* function to install the driver.

2.2 Including the driver in VxWorks

How to include the device drive in the VxWorks system is described in the VxWorks and Tornado manuals.

2.3 Hardware and system dependencies

The TPMC821 can be mounted to different hardware. This will sometimes need some hardware dependant adaptation.

PCI Initialization

The hardware must be configured before starting the driver. The following points must be guaranteed:

- The PCI spaces of the TPMC821 (PLX9050) must be set up to unused PCI areas. Memory and I/O accesses must be enabled in the PCI configuration space (see example below).
- The PCI interrupts must be set up (Interrupt routing and handler).

BSP dependencies

The *tpmc_pci.c* file has to be adapted, because there are some hardware and system dependent values (only PowerPC targets). Please check the following values:

- PCI_MEM_BRIDGE_OFFSET — This must be set to the offset, which is added by the PCI bridge (refer to BSP) when accessing PCI memory.
- PCI_IO_BRIDGE_OFFSET — This must be set to the offset, which is added by the PCI bridge (refer to BSP) when accessing PCI I/O spaces.
- int_dev_no (array) — This array defines the interrupt vectors/levels for #INTA of the different device position (first index = bus number, second index = device number). These values or the size of the table have to be adapted (if using busses with higher bus numbers). The interrupt vectors/levels depend on BSP.

Time factor

A counter constant is the last thing, which has to be configured. This constant is used for a delay, which is needed by the INTERBUS G4 controller in synchronous mode. The constant and a wait macro (waiting 5µs) are placed in the file *tpmctime.h*. There is a little function *tp821_TestTime()* in the example code, which will help to find the right constant. This value should always be calibrated when using a synchronous operation mode. It is not necessary for asynchronous operation modes.

The device driver uses this delay during the interrupt function. This may delay other tasks and interrupts (the times will be < 100µs).

2.4 Special installation for Intel x86 based targets

The TPMC821 device driver is fully adapted for Intel x86 based targets. This is done by conditional compilation directives inside the source code and controlled by the VxWorks global defined macro ***CPU***. If the contents of this macro are equal to *I80386*, *I80386* or *PENTIUM* special Intel x86 conforming code and function calls will be included.

The second problem for Intel x86 based platforms can't be solved by conditional compilation directives. Due to the fact that some Intel x86 BSP's doesn't map PCI memory spaces of devices which are not used by the BSP, the required CAN controller device registers can't be accessed.

To solve this problem a MMU mapping entry has to be added for the required TPMC821 PCI memory spaces prior the MMU initialization (*usrMmuInit()*) is done.

Please examine the BSP documentation or contact the BSP Vendor whether the BSP perform automatic PCI and MMU configuration or not. If the PCI and MMU initialization is done by the BSP the function *tp821PciInit()* won't be included and the user can skip to the following steps.

The C source file **tp821pci.c** contains the function *tp821PciInit().* This routine finds out all TPMC821 devices and adds MMU mapping entries for all used PCI memory spaces. Please insert a call to this function after the PCI initialization is done and prior to MMU initialization (*usrMmuInit()*).

If the Tornado 2.0 project facility is used, the right place to call the function *tp821PciInit()* is at the end of the function *sysHwInit()* in **sysLib.c** (can be opened from the project *Files* window).

If Tornado 1.0.1 compatibility tools are used insert the call to *tp821PciInit()* at the beginning of the root task (*usrRoot()*) in **usrConfig.c**.

Be sure that the function is called prior to MMU initialization otherwise the TPCM821 PCI spaces remains unmapped and an access fault occurs during driver initialization.

Please insert the following call at a suitable place in either **sysLib.c** or **usrConfig.c**:

```
tp821PciInit();
```

To link the driver object modules to VxWorks, simply add all necessary driver files to the project. If Tornado 1.0.1 *Standard BSP Builds...* is used add the object modules to the macro *MACH_EXTRA* inside the BSP Makefile (MACH_EXTRA = tp821drv.o tp821pci.o ...).

The function *tp821PciInit()* was designed for and tested on generic Pentium targets. If another BSP is used, please refer to BSP documentation or contact the technical support for required adaptation.

If strange errors appeared after system startup with the new build system please carrying out a VxWorks *build clean* and *build all*.

3 I/O system functions

This chapter describes the driver-level interface to the I/O system. The purpose of these functions is to install the driver in the I/O system, add and initialize devices.

3.1 tp821Drv()

NAME

tp821Drv() - installs the TPMC821 driver in the I/O system.

SYNOPSIS

void tp821Drv(void)

DESCRIPTION

This function installs the TPMC821 driver in the I/O system.

The call of this function is the first thing the user has to do before adding any device to the system or performing any I/O request.

RETURNS

OK or ERROR (if the driver cannot be installed)

INCLUDE FILES

tpmc821.h

3.2 tp821DevCreate()

NAME

tp821DevCreate() - adds TPMC821 device to the system and initializes device hardware.

SYNOPSIS

```
STATUS tp821DevCreate
    (
    char        *name,      /* name of the device to create                    */
    int         busNo,      /* bus number where the module is moutned          */
    int         deviceNo,   /* device number where the module is mounted       */
    int         functionNo  /* function number, must be always '0'             */
    )
```

DESCRIPTION

This routine is called to add a device to the system that will be serviced by the TPMC821 driver. This function must be called before performing any I/O request to this driver.

There are several device dependent arguments required for the device initialization and allocation of the system resources.

PARAMETER

The argument **name** specifies the name, which will select the device in future calls.

The arguments **busNo** and **deviceNo** specify the position of the TPMC821. These values are system dependent (refer to the carrier manual).

The argument **functionNo** must be left '0'. This value selects the module function. The TPMC821 supports only one function.

EXAMPLE

```
#include "tpmc821.h"

...

int status;

/*-------------------------------------------------------------
  Create a device "/tp821A"
  to select TPMC821 is mounted to bus 0 and device 16
  -------------------------------------------------------------*/
status = tp821DevCreate("/tp821A", 0, 16, 0);

...
```

RETURNS

OK or ERROR

INCLUDE FILES

tpmc821.h

4 I/O interface functions

This chapter describes the interface to the basic I/O system used for communication over the INTERBUS.

4.1 open()

NAME

open() - opens a device or file.

SYNOPSIS

```
int open
        (
        const char  *name,          /* name of the device to open                  */
        int         flags,          /* not used for TPMC821 driver, must be '0'  */
        int         mode            /* not used for TPMC821 driver, must be '0'  */
        )
```

DESCRIPTION

Before I/O can be performed to the TPMC821 device, a file descriptor must be opened by invoking the basic I/O function *open().*

PARAMETER

The parameter **name** selects the device which shall be opened.

The parameters **flags** and **mode** are not used and must be 0.

EXAMPLE

```
...

/*-------------------------------------------------------------
  Open the device named "/tpmc821A" for I/O
  -------------------------------------------------------------*/
fd = open("/tpmc821A", 0, 0);

...
```

RETURNS

A device descriptor number or ERROR (if the device does not exist or no device descriptors are available)

INCLUDE FILES

vxworks.h

tpmc821.h

SEE ALSO

ioLib, basic I/O routine - *open()*

4.2 read()

NAME

read() – reads data from the specified TPMC821 device.

SYNOPSIS

```
int read
        (
        int         fd,         /* device descriptor from opened TPMC821 device    */
        char        *buffer,    /* pointer to the data buffer                      */
        size_t      maxbytes    /* not used                                        */
        )
```

PARAMETER

The parameter **fd** is a file descriptor specifying the device which shall be used.

The argument **buffer** points to a driver-specific I/O parameter block. This buffer is segmented into parts with the data structure of the type *TP821_RW_SEGMENT* (see below).

The parameter **maxbytes** is not used by the TPMC821 Device Driver.

data structure *TP821_RW_SEGMENT*

```
typedef struct
{
unsigned short    itemNumber;  /* number of items (bytes, w..)      */
unsigned short    itemType;    /* TP821_BYTE|TP821_WORD|..          */
unsigned short    dataOffset;  /* Byte Off. in DATA IN/OUT reg      */
union
        {
        unsigned char       byte[1];
        unsigned short      word[1];
        unsigned long       lword[1];
        } u;
} TP821_RW_SEGMENT;
```

The argument **itemNumber** specifies how many elements of the specified type will follow.

The **itemType** specifies the length of the data element. Allowed values are:

TP821_END	Specifies the last segment of a segment list for data commands, no data follows.
TP821_BYTE	Specifies a segment with byte data. The union part **byte** will be used (Datalength = itemNumber * 1 byte).
TP821_WORD	Specifies a segment with word data. The union part **word** will be used (Datalength = itemNumber * 2 byte).
TP821_LWORD	Specifies a segment with longword data. The union part **lword** will be used (Datalength = itemNumber * 4 byte).

The argument **dataOffset** specifies the offset in the data area of the TPMC821. The specified data will be read from the data in base address + dataOffset (in byte).

The union **u** marks the first element of the data area of the segment. The area size is not specified by this array. It is specified with the **itemNumber** argument.

The data structure *TP821_RW_SEGMENT* will be put over the data buffer.

There are two MACROS defined in *tpmc821.h*, which will help setting up the data buffer.

The 1st function *SEGMENT_SIZE(pSeg)* calculates the length of the data segment. The data segment must be specified with the segment pointer in *pSeg*.

The 2nd function *NEXT_SEGMENT(pSeg)* calculates the start of the next segment. The actual data segment must be specified with the segment pointer in *pSeg*. The new data pointer will be the return value (see example below).

Example

The transmitted data shall be split into two segments and an end segment. The 1st segment shall have a size of 8 bytes, the 2nd segment shall have a size of 2 longwords. The contents of the 1st segment shall be read from data offset 8 and the 2nd segment shall be read from position 0. The data buffer segmentation will have the following layout.

Segment values (before calling the read function):

<u>1st segment:</u>

itemNumber:	8
itemType:	TP821_BYTE
itemOffset:	0x008
data:	(8 byte)

<u>2nd segment:</u>

itemNumber:	2
itemType:	TP821_LWORD
itemOffset:	0x000
data:	(2 longwords)

<u>End segment:</u>

itemNumber:	0
itemType:	TP821_END
itemOffset:	0x000
data:	(none)

The data buffer has the following layout (before calling the read function):

Offset	+0	+1	+2	+3	+4	+5	+6	+7
+0x00	0x00	0x08	0x00	0x01	0x00	0x08	*xx*	*xx*
+0x08	*xx*	*xx*	*xx*	*xx*	*xx*	*xx*	0x00	0x02
+0x10	0x00	0x04	0x00	0x00	*xx*	*xx*	*xx*	*xx*
+0x18	*xx*	*xx*	*xx*	*xx*	0x00	0x00	0x00	0x00
+0x20	0x00	0x00	*xx*	*xx*	*xx*	*xx*	*xx*	*xx*

The data input area of the TPMC821:

Offset	+0	+1	+2	+3	+4	+5	+6	+7
+0x00	0x12	0x34	0x56	0x78	0x9A	0xBC	0xDE	0xF0
+0x08	0x00	0x11	0x22	0x33	0x44	0x55	0x66	0x77

The data buffer has the following layout (after calling the read function):

Offset	+0	+1	+2	+3	+4	+5	+6	+7
+0x00	0x00	0x08	0x00	0x01	0x00	0x08	*0x00*	*0x11*
+0x08	*0x22*	*0x33*	*0x44*	*0x55*	*0x66*	*0x77*	0x00	0x02
+0x10	0x00	0x04	0x00	0x00	*0x12*	*0x34*	*0x56*	*0x78*
+0x18	*0x9A*	*0xBC*	*0xDE*	*0xF0*	0x00	0x00	0x00	0x00
+0x20	0x00	0x00	*xx*	*xx*	*xx*	*xx*	*xx*	*xx*

Segment values (after calling the read function):

<u>1st segment:</u>

itemNumber: 8

itemType: TP821_BYTE

itemOffset: 0x008

data: 0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77

<u>2nd segment:</u>

itemNumber: 2

itemType: TP821_LWORD

itemOffset: 0x000

data: 0x12345678, 0x9ABCDEF0

<u>End segment:</u>

itemNumber: 0

itemType: TP821_END

itemOffset: 0x000

data: (none)

EXAMPLE

```
#include "tpmc821.h"
...
unsigned char      segmentBuffer[100];
TP821_RW_SEGMENT   *pSeg;
unsigned long      result;
int                size;
/*-------------------------------------------------------------
  Read data from an open TPMC821 device,
  read data from offset 0,
  read the same data with byte, word and longword length,
  the length shall always be 4 byte
  -------------------------------------------------------------*/
size = 0; /* Checking buffer overrun (size always < 100) */

/* pointer to the first segment */
pSeg = (TP821_RW_SEGMENT*)&segmentBuffer;
pSeg->itemType     = TP821_BYTE;
pSeg->itemNumber   = 4;
pSeg->dataOffset   = 0;
size += SEGMENT_SIZE(pSeg);

/* same data read as word */
pSeg = NEXT_SEGMENT(pSeg);
pSeg->itemType     = TP821_WORD;
pSeg->itemNumber   = 2;
pSeg->dataOffset   = 0;
size += SEGMENT_SIZE(pSeg);

/* same data read as longword */
pSeg = NEXT_SEGMENT(pSeg);
pSeg->itemType     = TP821_LWORD;
pSeg->itemNumber   = 1;
pSeg->dataOffset   = 0;
size += SEGMENT_SIZE(pSeg);

/* End segment */
pSeg = NEXT_SEGMENT(pSeg);
pSeg->itemType     = TP821_END;
pSeg->itemNumber   = 0;
pSeg->dataOffset   = 0;
size += SEGMENT_SIZE(pSeg);

result = read(tp821_dev, &segmentBuffer, size);
if (result != ERROR)
{
     /* read successfully completed */
}
```

```
else
{
     /* read failed */;
}

...
```

RETURNS

ERROR or length of data buffer

INCLUDES

vxworks.h

tpmc821.h

SEE ALSO

ioLib, basic I/O routine - *read()*

4.3 write()

NAME

write() – writes data to the specified TPMC821 device.

SYNOPSIS

```
int write
        (
        int         fd,         /* device descriptor from opened TPMC821 device    */
        char        *buffer,    /* pointer to the data buffer                      */
        size_t      bytes       /* not used                                        */
        )
```

PARAMETER

The parameter **fd** is a file descriptor specifying the device which shall be used.

The argument **buffer** points to a driver-specific I/O parameter block. This data structure is named *TP821_RW_SEGMENT* (see below).

The parameter **bytes** is not used by the TPMC821 Device Driver.

data structure *TP821_RW_SEGMENT*

```
typedef struct
{
unsigned short    itemNumber;  /* number of items (bytes, w..)      */
unsigned short    itemType;    /* TP821_BYTE|TP821_WORD|..          */
unsigned short    dataOffset;  /* Byte Off. in DATA IN/OUT reg      */
union
        {
        unsigned char       byte[1];
        unsigned short      word[1];
        unsigned long       lword[1];
        } u;
} TP821_RW_SEGMENT;
```

The argument **itemNumber** specifies how many elements of the specified type will follow.

The **itemType** specifies the length of the data element. Allowed values are:

TP821_END	Specifies the last segment of a segment list for data commands, no data follows.
TP821_BYTE	Specifies a segment with byte data. The union part **byte** will be used (Datalength = itemNumber * 1 byte).
TP821_WORD	Specifies a segment with word data. The union part **word** will be used (Datalength = itemNumber * 2 byte).
TP821_LWORD	Specifies a segment with longword data. The union part **lword** will be used (Datalength = itemNumber * 4 byte).

The argument **dataOffset** specifies the offset in the data area of the TPMC821. The specified data will be written to the data in base address + dataOffset (in byte).

The union **u** marks the first element of the data area of the segment. The area size is not specified by this array. It is specified with the **itemNumber** argument.

The data structure *TP821_RW_SEGMENT* will be put over the data buffer.

There are two MACROS defined in *tpmc821.h*, which will help setting up the data buffer.

The 1st function *SEGMENT_SIZE(pSeg)* calculates the length of the data segment. The data segment must be specified with the segment pointer in *pSeg*.

The 2nd function *NEXT_SEGMENT(pSeg)* calculates the start of the next segment. The actual data segment must be specified with the segment pointer in *pSeg*. The new data pointer will be the return value (see example below).

Example

There are two data segments that shall be transmitted. The 1st segment has a size of 8 bytes, the 2nd segment shall have a size of 2 longwords. The contents of the 1st segment shall be written to data offset 8 and the 2nd segment shall be written to position 0. The data buffer segmentation will have the following layout.

Segment values:

<u>1st segment:</u>

itemNumber:	8
itemType:	TP821_BYTE
itemOffset:	0x008
data:	0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77

<u>2nd segment:</u>

itemNumber:	2
itemType:	TP821_LWORD
itemOffset:	0x000
data:	0x12345678, 0x9ABCDEF0

<u>End segment:</u>

itemNumber:	0
itemType:	TP821_END
itemOffset:	0x000
data:	(none)

The data buffer has the following layout:

Offset	+0	+1	+2	+3	+4	+5	+6	+7
+0x00	0x00	0x08	0x00	0x01	0x00	0x08	0x00	0x11
+0x08	0x22	0x33	0x44	0x55	0x66	0x77	0x00	0x02
+0x10	0x00	0x04	0x00	0x00	0x12	0x34	0x56	0x78
+0x18	0x9A	0xBC	0xDE	0xF0	0x00	0x00	0x00	0x00
+0x20	0x00	0x00	*xx*	*xx*	*xx*	*xx*	*xx*	*xx*

The data output area of the TPMC821 (after writing):

Offset	+0	+1	+2	+3	+4	+5	+6	+7
+0x00	0x12	0x34	0x56	0x78	0x9A	0xBC	0xDE	0xF0
+0x08	0x00	0x11	0x22	0x33	0x44	0x55	0x66	0x77

EXAMPLE

```
#include "tpmc821.h"

...
unsigned char      segmentBuffer[100];
TP821_RW_SEGMENT   *pSeg;
unsigned long      result;
int                size;

/*-------------------------------------------------------------
  Write data to an open TPMC821 device,
  write data to offset 0,
  write one word with data (0x1234)
  -------------------------------------------------------------*/
size = 0;     /* Checking buffer overrun (size always < 100) */

/* pointer to the first segment */
pSeg = (TP821_RW_SEGMENT*)&segmentBuffer;
pSeg->itemType     = TP821_WORD;
pSeg->itemNumber   = 1;
pSeg->dataOffset   = 0;
pSeg->u.word[0]    = 0x1234;
size += SEGMENT_SIZE(pSeg);

/* End segment */
pSeg = NEXT_SEGMENT(pSeg);
pSeg->itemType     = TP821_END;
pSeg->itemNumber   = 0;
pSeg->dataOffset   = 0;
size += SEGMENT_SIZE(pSeg);
```

```
result = write(tp821_dev[device], &segmentBuffer, size);
if (result != ERROR)
{
     /* write successfully completed */
}
else
{
     /* write failed */;
}

...
```

RETURNS

ERROR or length of data buffer

INCLUDE FILES

vxworks.h

tpmc821.h

SEE ALSO

ioLib, basic I/O routine - *write()*

4.4 ioctl()

NAME

ioctl() - performs an I/O control function.

SYNOPSIS

```
int ioctl
(
  int       fd,         /* device descriptor from opened TPMC821 device    */
  int       request,    /* select of control function    */
  int       arg         /* parameter buffer              */
)
```

DESCRIPTION

Special I/O operation that do not fit to the standard basic I/O calls (read, write) will be performed by calling the *ioctl()* function.

PARAMETER

The parameter **fd** specifies the device descriptor of the opened TPMC821 device.

The parameter **request** specifies the function which shall be executed.

The structure **arg** depends on the selected request (see description below).

RETURNS

OK or ERROR (if an error occurred)

INCLUDE FILES

vxworks.h

tpmc821.h

SEE ALSO

ioLib, basic I/O routine - *ioctl()*

4.4.1 FIO_TP821_BIT_CMD

This function code is used to execute a bit command. The bit command starts and executes a standard function. These functions and command bits are defined by the INTERBUS Master Firmware.

The request dependent argument **arg** points to a union named *TP821_CNTRL_STRUCT*.

data union *TP821_CNTRL_STRUCT*:

```
typedef union
{
  TP821_IOC_BCMD_STRUCT      bcmd;   /* FIO_TP821_BIT_CMD        */
  TP821_IOC_MBX_STRUCT       mbx;    /* FIO_TP821_MBX_WAIT       */
                                     /* FIO_TP821_MBX_NOWAIT     */
  TP821_IOC_DIAG_STRUCT      diag;   /* FIO_TP821_GET_DIAG       */
  TP821_IOC_CONFIG_STRUCT    config; /* FIO_TP821_CONFIGURE      */
} TP821_CNTRL_STRUCT;
```

For this function code the **bcmd** option is used. **bcmd** is a data structure named *TP821_IOC_BCMD_STRUCT*.

data structure *TP821_IOC_BCMD_STRUCT*:

```
typedef struct
{
      int              cmdBit;          /* Command bit (0..13)   */
      unsigned short   cmdParam;        /* Command parameter     */
} TP821_IOC_BCMD_STRUCT;
```

The argument **cmdBit** specifies the command bit.

The parameter for the command bit operation is specified in the **cmdParam** argument.

More information about the command bits and the parameter values can be found in the User Manuals for the INTERBUS Generation 4 which is parts of the TPMC821-DOC Engineering Documentation.

EXAMPLE

```
#include <vxWorks.h>
#include "tpmc821.h"

...

STATUS                  result;
TP821_CNTRL_STRUCT      cntrlBuf;

/*-------------------------------------------------------------
  Call standard function with a bit command,
  start data transfer (bit 0), (no parameter)
  -------------------------------------------------------------*/
cntrlBuf.bcmd.cmdBit =  (1 << 0);    /* use command bit 0    */
cntrlBuf.bcmd.cmdParam =    0;       /* parameter not used   */
result = ioctl(tp821_dev, FIO_TP821_BIT_CMD, &cntrlBuf);
if (result != ERROR)
{
     /* Bit command successfully executed */
}
else
{
     /* ERROR while execution */
}

...
```

4.4.2 FIO_TP821_MBX_WAIT

This function executes a mailbox command on the specified TPMC821 and waits for completion and a result will be returned.

The request dependent argument arg points to a union named *TP821_CNTRL_STRUCT*.

data union *TP821_CNTRL_STRUCT*:

```
typedef union
{
  TP821_IOC_BCMD_STRUCT        bcmd;        /* FIO_TP821_BIT_CMD        */
  TP821_IOC_MBX_STRUCT         mbx;         /* FIO_TP821_MBX_WAIT       */
                                            /* FIO_TP821_MBX_NOWAIT     */
  TP821_IOC_DIAG_STRUCT        diag;        /* FIO_TP821_GET_DIAG       */
  TP821_IOC_CONFIG_STRUCT      config;      /* FIO_TP821_CONFIGURE      */
} TP821_CNTRL_STRUCT;
```

For this function code the **mbx** option is used. **mbx** is a data structure named *TP821_IOC_MBX_STRUCT*.

data structure *TP821_IOC_MBX_STRUCT*:

```
typedef struct
{
      int                          cmdSize;     /* Command size in words      */
      unsigned short*              cmdBuffer;   /* Pointer to parameter buffer */
      int                          resultSize;  /* Result size in words    */
      unsigned short*              resultBuffer; /* Pointer to result buffer */
} TP821_IOC_MBX_STRUCT;
```

The argument **cmdSize** specifies the length of the command buffer **cmdBuffer**, which will be transmitted to the TPMC821.

The **resultSize** argument must specify the maximal length of result buffer **ResultBuffer**. When calling the function, after execution it returns the valid length of the **resultBuffer**.

More information about the mailbox commands and the parameters can be found in the User Manuals for the INTERBUS Generation 4 which is parts of the TPMC821-ED Engineering Documentation.

EXAMPLE

```
#include  <vxWorks.h>
#include  “tpmc821.h”

...

STATUS              result;
unsigned short      RequestPar[100];
unsigned short      ResultPar[100];
TP821_CNTRL_STRUCT cntrlBuf;

/*-------------------------------------------------------------
  Make a mailbox command and wait for completion,
  create a configuration service
  -------------------------------------------------------------*/
RequestPar[0] =    0x0710;   /* Create Configuration Service */
RequestPar[1] =    1;        /* 1 parameter follow     */
RequestPar[2] =    1;        /* number of frames to generate */

/* 3 words used in RequestPar */
cntrlBuf.mbx.cmdSize    = 3;
cntrlBuf.mbx.cmdBuffer  = RequestPar;

/* max. Size of ResultPar 100 */
cntrlBuf.mbx.resultSize      = 100;
cntrlBuf.mbx.resultBuffer    = ResultPar;

result = ioctl(tp821_dev, FIO_TP821_MBX_WAIT, &cntrlBuf);
if (result != ERROR)
{
     /* Mailbox command successfully executed */
}
else
{
     /* ERROR while execution */
}

...
```

4.4.3 FIO_TP821_MBX_NOWAIT

This function executes a mailbox command on the specified TPMC821 and do not wait for completion.

The request dependent argument **arg** points to a union named *TP821_CNTRL_STRUCT*.

data union *TP821_CNTRL_STRUCT*:

```
typedef union
{
    TP821_IOC_BCMD_STRUCT     bcmd;        /* FIO_TP821_BIT_CMD      */
    TP821_IOC_MBX_STRUCT      mbx;         /* FIO_TP821_MBX_WAIT     */
                                           /* FIO_TP821_MBX_NOWAIT   */
    TP821_IOC_DIAG_STRUCT     diag;        /* FIO_TP821_GET_DIAG     */
    TP821_IOC_CONFIG_STRUCT   config;      /* FIO_TP821_CONFIGURE    */
} TP821_CNTRL_STRUCT;
```

For this function code the **mbx** option is used. **mbx** is a data structure named *TP821_IOC_MBX_STRUCT*.

data structure *TP821_IOC_MBX_STRUCT*:

```
typedef struct
{
    int                 cmdSize;       /* Command size in words      */
    unsigned short*     cmdBuffer;     /* Pointer to parameter buffer */
    int                 resultSize;    /* Result size in words    */
    unsigned short*     resultBuffer;  /* Pointer to result buffer */
} TP821_IOC_MBX_STRUCT;
```

The argument **cmdSize** specifies the length of the command buffer **cmdBuffer**, which will be transmitted to the TPMC821.

The argument **resultSize** and **resultBuffer** are not used by this function.

More information about the mailbox commands and the parameters can be found in the User Manuals for the INTERBUS Generation 4 which is parts of the TPMC821-ED Engineering Documentation.

EXAMPLE

```
#include <vxWorks.h>
#include "tpmc821.h"

...

STATUS              result;
unsigned short      RequestPar[100];
TP821_CNTRL_STRUCT cntrlBuf;

/*-------------------------------------------------------------
  Make a mailbox command and wait for completion,
  reset controller Board (make cold start)
  -------------------------------------------------------------*/
RequestPar[0] =    0x0956;   /* Reset Controller Board Service     */
RequestPar[1] =    1;        /* 1 parameter follow     */
RequestPar[2] =    0;        /* Cold start */

/* 3 words used in RequestPar */
cntrlBuf.mbx.cmdSize    = 3;
cntrlBuf.mbx.cmdBuffer  = RequestPar;

/* no result Parameter */
cntrlBuf.mbx.resultSize     = 0;
cntrlBuf.mbx.resultBuffer   = NULL;

result = ioctl(tp821_dev, FIO_TP821_MBX_NOWAIT, &cntrlBuf);
if (result != ERROR)
{
     /* Mailbox command successfully started */
}
else
{
     /* ERROR while execution */
}

...
```

4.4.4 FIO_TP821_GET_DIAG

This function returns diagnostic information from the specified TPMC821.

The request dependent argument **arg** points to a union named *TP821_CNTRL_STRUCT*.

data union *TP821_CNTRL_STRUC*':

```
typedef union
{
    TP821_IOC_BCMD_STRUCT      bcmd;        /* FIO_TP821_BIT_CMD        */
    TP821_IOC_MBX_STRUCT       mbx;         /* FIO_TP821_MBX_WAIT       */
                                            /* FIO_TP821_MBX_NOWAIT     */
    TP821_IOC_DIAG_STRUCT      diag;        /* FIO_TP821_GET_DIAG       */
    TP821_IOC_CONFIG_STRUCT    config;      /* FIO_TP821_CONFIGURE      */
} TP821_CNTRL_STRUCT;
```

For this function code the **diag** option is used. **diag** is a data structure named *TP821_IOC_DIAG_STRUCT*.

data structure *TP821_IOC_DIAG_STRUCT*:

```
typedef struct
{
    unsigned short     sysfailReg;     /* contents of Sysfail Register     */
    unsigned short     configReg;      /* contents of Config Register      */
    unsigned short     diagReg;        /* contents of Diag. Register       */
    unsigned char      hardwareFail;   /* HW failure has been detected     */
    unsigned char      initComplete;   /* HW init has completed with success */
} TP821_IOC_DIAG_STRUCT;
```

The returned values of **sysfailReg**, **configReg** and **diagReg** are the actual values of the corresponding hardware registers Status Sysfail Register, Configuration Register and Master Diagnostic Status Register. Information about these registers and their flags can be found in the User Manuals for the INTERBUS Generation 4 which is parts of the TPMC821-ED Engineering Documentation.

The **hardwareFail** argument returns *TRUE* if a hardware failure occurred or *FALSE* if no hardware failure occurred.

The **initComplete** argument returns *TRUE* if the INTERBUS firmware has completed initialization. If it is still initializing *FALSE* value will be returned.

EXAMPLE

```
#include <vxWorks.h>
#include “tpmc821.h”

...

STATUS             result;
TP821_CNTRL_STRUCT cntrlBuf;

/*-------------------------------------------------------------
  Read diagnostic values from the specified device
  -------------------------------------------------------------*/
result = ioctl(tp821_dev, FIO_TP821_GET_DIAG, &cntrlBuf);
if (result != ERROR)
{
     /* diagnostic values successfully read */
     printf("Status Sysfail Register    : %04Xh\n",
          cntrlBuf.diag.sysfailReg);
     printf("Configuration Register     : %04Xh\n",
          cntrlBuf.diag.configReg);
     printf("Master Diagnostic Register : %04Xh\n",
          cntrlBuf.diag.diagReg);
     printf("Hardware Failure           : %s\n",
          cntrlBuf.diag.hardwareFail ? "TRUE" : "FALSE" );
     printf("Initialization done        : %s\n",
          cntrlBuf.diag.initComplete ? "TRUE" : "FALSE" );
}
else
{
     /* ERROR while execution */
}
```

4.4.5 FIO_TP821_CONFIGURE

This function can be used to announce a changing of the operation mode to the driver and to change the timeout values for mailbox and data accesses.

The request dependent argument **arg** points to a union named *TP821_CNTRL_STRUCT*.

data union *TP821_CNTRL_STRUCT*:

```
typedef union
{
    TP821_IOC_BCMD_STRUCT     bcmd;      /* FIO_TP821_BIT_CMD       */
    TP821_IOC_MBX_STRUCT      mbx;       /* FIO_TP821_MBX_WAIT      */
                                         /* FIO_TP821_MBX_NOWAIT    */
    TP821_IOC_DIAG_STRUCT     diag;      /* FIO_TP821_GET_DIAG      */
    TP821_IOC_CONFIG_STRUCT   config;    /* FIO_TP821_CONFIGURE     */
} TP821_CNTRL_STRUCT;
```

For this function code the **config** option is used. **config** is a data structure named *TP821_IOC_CONFIG_STRUCT*.

data structure *TP821_IOC_CONFIG_STRUCT*:

```
typedef struct
{
    unsigned long     op_mode;      /* operation mode          */
    long              dt_timeout;   /* Data Timeout in seconds */
    long              mb_timeout;   /* Mailbox Timeout in seconds */
} TP821_IOC_CONFIG_STRUCT;
```

The argument **op_mode** announces the new operation mode. Detailed information about the operation modes and how to start them can be found in the User Manuals for the INTERBUS Generation 4 which is parts of the TPMC821-ED Engineering Documentation. Possible values are:

TP821_ASYNC	asynchronous operation mode (default)
TP821_ASYNC_LOCK	asynchronous operation mode with consistency locking
TP821_BUSSYNC	bus synchronous mode
TP821_PRGSYNC	program synchronous

The argument **dt_timout** specifies a new timeout value for data accesses. This value must be specified in seconds.

The argument **mb_timout** specifies a new timeout value for mailbox accesses. This value must be specified in seconds.

EXAMPLE

```
#include <vxWorks.h>
#include "tpmc821.h"

...

STATUS             result;
TP821_CNTRL_STRUCT cntrlBuf;

/*-------------------------------------------------------------
  Announce new operation mode (asynchronous without lock)
  data timeout:    10 seconds
  mailbox timeout:  3 seconds
  -------------------------------------------------------------*/
cntrlBuf.config.op_mode      =    TP821_ASYNC;
cntrlBuf.config.dt_timeout   =    10;
cntrlBuf.config.mb_timeout   =    3;
result = ioctl(tp821_dev, FIO_TP821_CONFIGURE, &cntrlBuf);
if (result != ERROR)
{
     /* Device successfully configured */
}
else
{
     /* ERROR while configuration */
}

...
```

4.4.6 FIO_TP821_SET_HOST_FAIL

This function sets the host interrupt request to announce a serious host system failure. How to use the host interrupt is described in the TIP821 User Manual and in the User Manuals for the INTERBUS Generation 4 which are parts of the TPMC821-ED Engineering Documentation.

The request dependent argument **arg** is not used for this function.

EXAMPLE

```
#include <vxWorks.h>
#include "tpmc821.h"

...

STATUS   result;

/*-------------------------------------------------------------
  Set host fail interrupt
  -------------------------------------------------------------*/
result = ioctl(tp821_dev, FIO_TP821_SET_HOST_FAIL, 0);
if (result != ERROR)
{
     /* setting host fail interrupt request succeeded */
}
else
{
     /* setting host fail interrupt request failed */
}

...
```

4.4.7 FIO_TP821_REMOVE_HOST_FAIL

This function removes the host interrupt request which announces a serious host system failure. How to use the host interrupt is described in the TIP821 User Manual and in the User Manuals for the INTERBUS Generation 4 which is parts of the TPMC821-ED Engineering Documentation.

The request dependent argument **arg** is not used for this function.

EXAMPLE

```
#include <vxWorks.h>
#include "tpmc821.h"

...

STATUS   result;

/*-------------------------------------------------------------
  Remove host fail interrupt
  -------------------------------------------------------------*/
result = ioctl(tp821_dev, FIO_TP821_REMOVE_HOST_FAIL, 0);
if (result != ERROR)
{
     /* removing host fail interrupt request succeeded */
}
else
{
     /* removing host fail interrupt request failed */
}

...
```

4.4.8 FIO_TP821_CLEAR_HWERROR

This function clears the hardware error flag, which is set on service interrupt requests generated on hardware failures of the INTERBUS Master. More information about the service interrupt request can be found in the TIP821 User Manual and in the User Manuals for the INTERBUS Generation 4 which is parts of the TPMC821-ED Engineering Documentation.

The request dependent argument **arg** is not used for this function.

EXAMPLE

```
#include <vxWorks.h>
#include “tpmc821.h”

...

STATUS   result;

/*-------------------------------------------------------------
  Reset hardware error flag
  -------------------------------------------------------------*/
result = ioctl(tp821_dev, FIO_TP821_CLEAR_HWERROR, 0);
if (result != ERROR)
{
     /* resetting hardware error succeeded */
}
else
{
     /* resetting hardware error failed */
}
...
```

5 Appendix

This chapter describes the symbols which are defined in the file *tpmc821.h*.

5.1 Predefined Symbols

Segment Types

TP821_END	0	Specify the last segment of a segment list for data commands
TP821_BYTE	1	Specify a segment with byte data
TP821_WORD	2	Specify a segment with word data
TP821_LWORD	4	Specify a segment with longword data

Operating Modes

TP821_ASYNC	1	Specify asynchronous operation mode without consistency locking
TP821_ASYNC_LOCK	2	Specify asynchronous operation mode with consistency locking
TP821_BUSSYNC	3	Specify bus synchronous operation mode
TP821_PRGSYNC	4	Specify program synchronous operation mode

Special Control Functions

FIO_TP821_BIT_CMD	110	Special function code selecting the bit command
FIO_TP821_MBX_WAIT	111	Special function code selecting the mailbox command waiting for a result
FIO_TP821_MBX_NOWAIT	112	Special function code selecting the mailbox command, not waiting for a result
FIO_TP821_GET_DIAG	113	Special function code selecting the diagnostic function, which reads the device state
FIO_TP821_CONFIGURE	114	Special function code selecting the function to configure the device parameter
FIO_TP821_SET_HOST_FAIL	115	Special function code to set the host fail interrupt request
FIO_TP821_REMOVE_HOST_FAIL	116	Special function code to removing the host fail interrupt request
FIO_TP821_CLEAR_HWERROR	117	Special function code for removing the hardware error flag, which disables data accesses

5.2 Additional Error Codes

If the device driver creates an error the error codes are stored in the *errno*. They can be read with the VxWorks function *errnoGet()* or *printErrno().*

S_tp821Drv_NXIO	0x08210001	There is no TPMC821 mounted to the specified location.
S_tp821Drv_ICMD	0x08210002	An illegal function code has been selected.
S_tp821Drv_MEMERR	0x08210003	Driver can not allocate memory.
S_tp821Drv_PARAERR	0x08210004	An illegal parameter value has been specified.
S_tp821Drv_DEVERR	0x08210005	A device I/O error occurred (TPMC821 is not ready).
S_tp821Drv_BUSY	0x08210006	Selected device is already busy.
S_tp821Drv_ILLBUFSIZE	0x08210007	Specified buffer size is too small.
S_tp821Drv_TIMEOUT	0x08210008	Request timed out.
S_tp821Drv_ILLBIT	0x08210009	An illegal bit has been specified.
S_tp821Drv_BUSSTOPPED	0x0821000A	The specified access is not working with a stopped INTERBUS.