
TPMC821-SW-92

QNX4 Device Driver

INTERBUS Master G4 PMC

Version 1.0.x

User Manual

Issue 1.0

November 2003

TPMC821-SW-92

INTERBUS Master G4 PMC

QNX4 Device Driver

This document contains information, which is proprietary to TEWS TECHNOLOGIES GmbH. Any reproduction without written permission is forbidden.

TEWS TECHNOLOGIES GmbH has made any effort to ensure that this manual is accurate and complete. However TEWS TECHNOLOGIES GmbH reserves the right to change the product described in this document at any time without notice.

TEWS TECHNOLOGIES GmbH is not liable for any damage arising out of the application or use of the device described herein.

©2003 by TEWS TECHNOLOGIES GmbH

Issue	Description	Date
1.0	First Issue	November 15, 2003

Table of Content

1	INTRODUCTION.....	4
2	INSTALLATION.....	5
	2.1 Build the device driver	5
	2.2 Start the driver process.....	5
3	DRIVER INTERFACE	7
	3.1 Find out the server process id.....	7
	3.2 Send I/O request.....	8
	3.2.1 TPMC821_IN_BUFFER.....	8
	3.2.2 TPMC821_OUT_BUFFER.....	9
4	DRIVER FUNCTIONS.....	11
	4.1 TPMC821_READ.....	12
	4.2 TPMC821_WRITE	18
	4.3 TPMC821_BIT_CMD	23
	4.4 TPMC821_MBX_WAIT.....	25
	4.5 TPMC821_MBX_NOWAIT	28
	4.6 TPMC821_GET_DIAG.....	30
	4.7 TPMC821_CONFIGURE	32
	4.8 TPMC821_SET_HOST_FAIL.....	34
	4.9 TPMC821_REMOVE_HOST_FAIL	36
	4.10TPMC821_CLEAR_HWERROR	38

1 Introduction

The TPMC821-SW-92 QNX4 device driver allows the operation of a TPMC821 – INTERBUS Master PMC on QNX4 operating systems with Intel or Intel-compatible x86 CPU.

The TPMC821 device driver is basically a user-level server program. The interactions between client process and the TPMC821 server process are realized via messages.

To start an I/O request the client process has to send an appropriate message, which contains a function code and optional parameter, to the server process. After the I/O operation has finished the server process replies the request message with a completion status and optional process data to caller.

The TPMC821 device driver includes the following functions:

- using all possible bus operation modes
 - asynchronous mode without consistency locking
 - asynchronous mode with consistency locking
 - bus synchronous mode
 - program synchronous mode
- using bit commands
- using mailbox commands
- reading data
- writing data
- controlling the host interrupt request
- resetting hardware error

2 Installation

The software is delivered on a PC formatted 3½" HD diskette.

Following files are located on the diskette:

tpmc821.c	Driver source code
tpmc821isr.c	Drivers interrupt functions
tpmc821.h	Definitions and data structures for driver and application
tpmc821Def.h	Device driver include
tpxxxhwdep.c	Hardware interface functions
tpxxxhwdep.h	Device driver include for interface functions
Makefile	Device driver make file
/example/example.c	Example application
TPMC821-SW-92.pdf	This Manual in PDF format

For installation copy these files into a desired target directory.

2.1 Build the device driver

1. Change to the target directory
2. Execute the Makefile

```
# make
```

2.2 Start the driver process

As mentioned in the introduction of this reference manual the TPMC821 device driver is implemented as a user-level server process. To start this process you have to enter the process name with optional parameter from the command shell.

```
./tpmc821 &
```

Now the server process will start with default configuration that means the server process starts scanning the PCI-bus for TPMC821 modules and attaches the first TPMC821 found for this server process. The default process priority is 19.

If you have multiple TPMC821 installed in your system, you have to start a server process for each TPMC821. To distinguish between multiple TPMC821s on the PCI-bus you must start the server process with the argument `-I<index>`. This argument specifies an index for the search order on the PCI-bus. The first TPMC821 module found has index 0 the second index 1 and so on.

To start server processes for two TPMC821 you have to enter the following command lines:

```
./tpmc821 -I0 &
```

```
./tpmc821 -I1 &
```

If the default priority (19) isn't suitable, you can specify the process priority with the argument *-P<priority>*.

```
./tpmc821 -I0 -P10 &
```

The server process has to allocate buffers for command receive and reply. The default size of the buffers is 1KByte (0x400 Bytes). If the application needs greater data packages, this size can be modified with the argument *-B<size>*. *<size>* is specified in bytes.

```
./tpmc821 -B2048
```

3 Driver Interface

Basically all communication between the application process (client) and the TPMC821 device driver (server) is done via the QNX message system. An I/O request can be issued by sending a message with the *Send()* function to the appropriate server process. The server process performs all necessary processing for the requested command and finishes the request by sending a reply message to the client process.

To determine the necessary process id of the server process, the client program can invoke the *qnx_name_locate()* function. The server process has registered itself under a unique name (e.g. *tpmc821/0*).

Note that many system calls use data structures, which should be obtained in a program from appropriate header files. Necessary header files are listed with the system call synopsis.

The following section describes how you can determine the process id of the TPMC821 server process and how you can send I/O requests to the driver.

3.1 Find out the server process id

Before an application program can send I/O request messages, the process id of the appropriate server process must be known. Because every server process registers itself under a unique name we can use this name to locate the server process.

A valid server name contains the module name (*tpmc821*) and a zero based index separated by a slash (*/*). The index is equal to the index specified as argument on command line when the server process was started. If no *"/*" argument was specified the index is always 0.

A valid server name for the first device looks like this:

```
tpmc821/0
```

The corresponding process id (pid) can be retrieved with the *qnx_name_locate()* function (see also C Library Reference).

EXAMPLE:

```
#include <sys/name.h>

...

pid_t    pid;

pid =    qnx_name_locate(    0, "tpmc821/0", 0, NULL);

if (pid == -1) {
    /* server process not found */
}

...
```

3.2 Send I/O request

In order to request the driver to perform I/O processing, you have to send a well-defined message (*pointed to by smsg*) with the kernel function *Send()* to the server process. After sending the message the application process becomes blocked (reply blocked) until the server process is able to complete the request.

After completion the application task receives a reply message (*pointed to by rmsg*) from the server and leaves the blocked state. The received message contains the return status of the I/O operation and optional process data (e.g. input port data).

For a detailed description of the kernel function *Send()* please refer to the *QNX C Library Reference Manual*.

The structures of the send message (*TPMC821_IN_BUFFER*) and receive message (*TPMC821_OUT_BUFFER*) are defined in *tpmc821.h* and have the following layout:

3.2.1 TPMC821_IN_BUFFER

```
typedef struct
{
    int             ControlCode;
    long           timeout;
    union {
        TPMC821_RW_SEGMENT      rw;
        TPMC821_BCMD_STRUCT     bcmd;
        TPMC821_MBX_STRUCT      mbx;
        TPMC821_CONFIG_STRUCT   config;
    } u;
} TPMC821_IN_BUFFER;
```

ControlCode

Specifies the control command of the operation. This value identifies the specific operation to be performed. Valid values are defined in *tpmc821.h* and described in detail in the next chapter.

timeout

Specifies the command timeout. If the TPMC821 is executing a command, the new command will be queued until the TPMC821 is no longer busy or the specified timeout passed. This parameter is specified in seconds.

u.rw

Contains parameters for the read and write operation. See also *TPMC821_READ* and *TPMC821_WRITE* for detailed information.

u.bcmd

Contains parameters bit command operation. See also *TPMC821_BIT_CMD* for detailed information.

u.mbx

Contains parameters for the mailbox command operation. See also *TPMC821_MBX_WAIT* and *TPMC821_MBX_NOWAIT* for detailed information.

u.config

Contains parameters for the configuration operation. See also *TPMC821_CONFIGURE* for detailed information.

3.2.2 TPMC821_OUT_BUFFER

```
typedef struct
{
    int                ReturnStatus;
    union {
        TPMC821_RW_SEGMENT        rw;
        TPMC821_MBX_STRUCT        mbx;
        TPMC821_DIAG_STRUCT       diag;
    }                          u;
} TPMC821_OUT_BUFFER;
```

ReturnStatus

Returns the error/status code of the I/O operation. If the function succeeds, the value of *ReturnStatus* is 0. If the function fails, the value of *cmdStat* is set to an error code (see also *tpmc821.h*).

u.rw

Contains the data read from the TPMC821. For detailed information see also *TPMC821_READ*.

u.mbx

Returns data read from the TPMC821 when executing a mailbox command.

u.diag

Contains the returned diagnostic values. For detailed information see also *TPMC821_READ*.

EXAMPLE

```
#include <sys/kernel.h>
#include "tpmc821.h"

...

pid_t          pid;
int            result;
TPMC821_IN_BUFFER  smsg;    /* driver input parameter */
TPMC821_OUT_BUFFER rmsg;    /* driver output parameter */

smsg.ControlCode = TPMC821_GET_DIAG;

/*
** Send request to the device driver
*/
result = Send(pid, &smsg, &rmsg, sizeof(smsg), sizeof(rmsg));
if( result == 0 )
{
    if( rmsg.cmdStat == EOK )
    {
        printf("Init complete %s\n",
               rmsg.u.diag.initComplete ? "Yes" : "NO");
    }
    else
    {
        printf("Read diagnostic failed (%d)\n",
               rmsg.ReturnStatus);
    }
}
else
{
    printf( "Send failed --> Error = %d.\n", errno );
}

...
```

4 Driver Functions

To request the driver to perform the desired I/O operation the application process must send an appropriate message *TPMC821_IN_BUFFER* to the device driver (server process). The *TPMC821_IN_BUFFER* (smsg) contains a function code and optional parameter for the desired I/O operation.

The following functions are support by a TPMC821 driver:

Value	Meaning
<i>TPMC821_READ</i>	read data from the dual ported memory
<i>TPMC821_WRITE</i>	write data to the dual ported memory
<i>TPMC821_BIT_CMD</i>	execute a bit command
<i>TPMC821_MBX_WAIT</i>	execute a mailbox command, waiting for completion
<i>TPMC821_MBX_NOWAIT</i>	execute a mailbox command, return immediately after starting the command
<i>TPMC821_GET_DIAG</i>	get diagnostic values
<i>TPMC821_CONFIGURE</i>	configure TPMC821 mode, and timeouts
<i>TPMC821_SET_HOST_FAIL</i>	set host fail interrupt
<i>TPMC821_REMOVE_HOST_FAIL</i>	remove host fail interrupt
<i>TPMC821_CLEAR_HWERROR</i>	clear hardware error flag

See behind for more detailed information on each control code.

4.1 TPMC821_READ

This function reads data from the dual ported memory of the TPMC821.

The *command* code and *timeout* must be set in *TPMC821_IN_BUFFER*. The *rw* selection of union will be used for input parameters. (See below)

The *ReturnStatus* will be returned in *TPMC821_OUT_BUFFER*. The data read from the TPMC821 will be returned in the union selection *rw*. (See below)

The selection *rw* of the unions in *TPMC821_IN_BUFFER*, and *TPMC821_OUT_BUFFER* reserve space for only one data element. The application has to allocate a buffer where the whole read structure fits in.

```
typedef struct
{
    unsigned short    itemNumber; /* number of items (bytes, word, ...) */
    unsigned short    itemType;   /* TPMC821_BYTE, TPMC821_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;
} TPMC821_RW_SEGMENT;
```

itemNumber

Specifies how many elements of the specified type will follow in the data union.

itemType

Specifies the length of the data element. Allowed item types are:

Value	Meaning
TPMC821_END	Specifies the last segment of a segment list for data commands. No data follows.
TPMC821_BYTE	Specifies a segment with byte data. The union part <i>byte</i> will be used. (Datalength = itemNumber * 1 byte)
TPMC821_WORD	Specifies a segment with word data. The union part <i>word</i> will be used. (Datalength = itemNumber * 2 byte)
TPMC821_LWORD	Specifies a segment with longword data. The union part <i>lword</i> will be used. (Datalength = itemNumber * 4 byte)

dataOffset

Specifies the offset in the data area of the TPMC821. The offset is specified in bytes. The specified data will be read from baseaddress + dataOffset.

u

The union 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 MACRO '*SEGMENT_SIZE(pSeg)*' calculates the length of the data segment. The data segment must be specified with the segment pointer in pSeg.

The 2nd MACRO '*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)

And there are two other MACROS returning the buffer sizes for input and output buffer without the union.

The 1st MACRO '*BUFIN_SIZE_RAW*' returns the size of *TPMC821_IN_BUFFER* without the union.

The 2nd MACRO '*BUFOUT_SIZE_RAW*' returns the size of *TPMC821_OUT_BUFFER* without the union.

Example

The 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):

1st segment:

itemNumber: 8
 itemType: TPMC821_BYTE
 itemOffset: 0x8
 data: (8 Bytes)

2nd segment:

itemNumber: 2
 itemType: TPMC821_LWORD
 itemOffset: 0x0
 data: (2 longwords)

End segment:

itemNumber: 0
 itemType:
 itemOffset: 0x0
 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):

1st segment:

itemNumber: 8
 itemType: TPMC821_BYTE
 itemOffset: 0x8
 data: 0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77

2nd segment:

itemNumber: 2
 itemType: TPMC821_LWORD
 itemOffset: 0x0
 data: 0x12345678, 0x9ABCDEF0

End segment:

itemNumber: 0
 itemType: TPMC821_END
 itemOffset: 0x0
 data: (none)

EXAMPLE

```
#include <sys/kernel.h>
#include "tpmc821.h"

#define IO_BUF_SIZE 100

...

pid_t    pid;
int      result;
int      size;
TPMC821_IN_BUFFER *in_par; /* pointer to driver input parameter */
TPMC821_OUT_BUFFER *out_par; /* pointer to driver output parameter */
TPMC821_RW_SEGMENT *pSeg;

in_par = (TPMC821_IN_BUFFER*)malloc(IO_BUF_SIZE);    /* Get a buffer */
out_par = (TPMC821_OUT_BUFFER*)malloc(IO_BUF_SIZE); /* Get a buffer */
size = 0;

/* pointer to the first segment */
pSeg = (TPMC821_RW_SEGMENT*)&in_par->u.rw;
pSeg->itemType = TPMC821_BYTE;
pSeg->itemNumber = 8;
pSeg->dataOffset = 8;
size += SEGMENT_SIZE(pSeg);

/* Second segment */
pSeg = NEXT_SEGMENT(pSeg);
pSeg->itemType = TPMC821_LWORD;
pSeg->itemNumber = 2;
pSeg->dataOffset = 0;
size += SEGMENT_SIZE(pSeg);

/* End segment */
pSeg = NEXT_SEGMENT(pSeg);
pSeg->itemType = TPMC821_END;
pSeg->itemNumber = 0;
pSeg->dataOffset = 0;
size += SEGMENT_SIZE(pSeg);

...
```

```
...

/* Send request to the device driver */
in_par->ControlCode = TPMC821_READ;
in_par->timeout = 2; /* timeout after 2 seconds */
result = Send(pid, in_par, out_par, BUFIN_SIZE_RAW + size, IO_BUF_SIZE);
if( result == 0 )
{
    if(out_par.ReturnStatus == 0 )
    {
        printSegments(&out_par->u.rw);
    }
    else
    {
        printf("\nRead failed (%d)\n",
            out_par->ReturnStatus);
    }
}
else
{
    printf( "\nSend failed --> Error = %d.\n", errno );
}

...

/* This function prints out the received data */
static void printSegments
(
    TPMC821_RW_SEGMENT *pSeg
)
{
    int i;

    printf("\nSEGMENTS:\n");
    while(pSeg->itemType != TPMC821_END)
    {
        switch(pSeg->itemType)
        {
            case TPMC821_BYTE:
                for(i = 0; i < pSeg->itemNumber; i++)
                {
                    printf("%02X ", pSeg->u.byte[i]);
                }
                printf("\n");
                break;
        }
    }
}

...
```

```

...

case TPMC821_WORD:
    for(i = 0; i < pSeg->itemNumber; i++)
    {
        printf("%04X ", pSeg->u.word[i]);
    }
    printf("\n");
    break;

case TPMC821_LWORD:
    for(i = 0; i < pSeg->itemNumber; i++)
    {
        printf("%08lX ", pSeg->u.lword[i]);
    }
    printf("\n");
    break;
}

pSeg = NEXT_SEGMENT(pSeg);
}

printf("\n");
}

...

```

RETURNS

ReturnStatus	value	description
	0x00000000	OK
<i>TPMC821_ERR_PARAERR</i>	0x08210004	Illegal parameter value found
<i>TPMC821_ERR_DEVERR</i>	0x08210005	Hardware error or module not initialized
<i>TPMC821_ERR_TIMEOUT</i>	0x08210008	The command or an access timed out
<i>TPMC821_ERR_BUSSTOPPED</i>	0x0821000A	The bus is stopped
<i>TPMC821_ERR_NOMEM</i>	0x0821000F	The driver can not allocate enough memory

4.2 TPMC821_WRITE

This function writes data to the dual ported memory of the TPMC821.

The *command* code and *timeout* must be set in *TPMC821_IN_BUFFER*. The *rw* selection of union will be used for input parameters. (See below)

The *ReturnStatus* will be returned in *TPMC821_OUT_BUFFER*. No other values will be returned.

The selection *rw* of the union in *TPMC821_IN_BUFFER* reserves space for only one data element. The application has to allocate a buffer where the whole write structure fits in.

```
typedef struct
{
    unsigned short    itemNumber; /* number of items (bytes, word, ...) */
    unsigned short    itemType;   /* TPMC821_BYTE, TPMC821_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;
} TPMC821_RW_SEGMENT;
```

itemNumber

Specifies how many elements of the specified type will follow in the data union.

itemType

Specifies the length of the data element. Allowed item types are:

Value	Meaning
TPMC821_END	Specifies the last segment of a segment list for data commands. No data follows.
TPMC821_BYTE	Specifies a segment with byte data. The union part <i>byte</i> will be used. (Datalength = itemNumber * 1 byte)
TPMC821_WORD	Specifies a segment with word data. The union part <i>word</i> will be used. (Datalength = itemNumber * 2 byte)
TPMC821_LWORD	Specifies a segment with longword data. The union part <i>lword</i> will be used. (Datalength = itemNumber * 4 byte)

dataOffset

Specifies the offset in the data area of the TPMC821. The offset is specified in bytes. The specified data will be written to in baseaddress + dataOffset.

u

The union 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 MACRO 'SEGMENT_SIZE(pSeg)' calculates the length of the data segment. The data segment must be specified with the segment pointer in pSeg.

The 2nd MACRO '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)

And there are one other MACROS returning the buffer size for the input buffer without the union.

The MACRO 'BUFIN_SIZE_RAW' returns the size of *TPMC821_IN_BUFFER* without the union.

Example

The 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 write data at offset 8 and the 2nd segment shall write at position 0. The data buffer segmentation will have the following layout.

Segment values (before calling the read function):

1st segment:

itemNumber: 8
 itemType: TPMC821_BYTE
 itemOffset: 0x8
 data: (8 Bytes) 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08

2nd segment:

itemNumber: 2
 itemType: TPMC821_LWORD
 itemOffset: 0x0
 data: (2 longwords) 0x11223344, 0x55667788

End segment:

itemNumber: 0
 itemType:
 itemOffset: 0x0
 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	0x01	0x02
+0x08	0x03	0x04	0x05	0x06	0x07	0x08	0x00	0x02
+0x10	0x00	0x04	0x00	0x00	0x11	0x22	0x33	0x44
+0x18	0x55	0x66	0x77	0x88	0x00	0x00	0x00	0x00
+0x20	0x00	0x00	xx	xx	xx	xx	xx	xx

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

Offset	+0	+1	+2	+3	+4	+5	+6	+7
+0x00	0x11	0x22	0x33	0x44	0x55	0x66	0x77	0x88
+0x08	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08

EXAMPLE

```
#include <sys/kernel.h>
#include "tpmc821.h"

#define IO_BUF_SIZE 100

...

pid_t pid;
int result;
int size;
TPMC821_IN_BUFFER *in_par; /* pointer to driver input parameter */
TPMC821_OUT_BUFFER out_par; /* output parameter */
TPMC821_RW_SEGMENT *pSeg;

in_par = (TPMC821_IN_BUFFER*)malloc(IO_BUF_SIZE); /* Get a buffer */
size = 0;

/* pointer to the first segment */
pSeg = (TPMC821_RW_SEGMENT*)&in_par->u.rw;
pSeg->itemType = TPMC821_BYTE;
pSeg->itemNumber = 8;
pSeg->dataOffset = 8;
pSeg->u.byte[0] = 0x01;
pSeg->u.byte[1] = 0x02;
...
pSeg->u.byte[7] = 0x08;
size += SEGMENT_SIZE(pSeg);

...
```

```
...

/* Second segment */
pSeg = NEXT_SEGMENT(pSeg);
pSeg->itemType = TPMC821_LWORD;
pSeg->itemNumber = 2;
pSeg->dataOffset = 0;
pSeg->u.lword[0] = 0x11223344;
pSeg->u.lword[1] = 0x55667788;
size += SEGMENT_SIZE(pSeg);

/* End segment */
pSeg = NEXT_SEGMENT(pSeg);
pSeg->itemType = TPMC821_END;
pSeg->itemNumber = 0;
pSeg->dataOffset = 0;
size += SEGMENT_SIZE(pSeg);

/* Send request to the device driver */
in_par->ControlCode = TPMC821_WRITE;
in_par->timeout = 2; /* timeout after 2 seconds */
result = Send(pid, in_par, &out_par,
              BUFIN_SIZE_RAW + size, sizeof(TPMC821_OUT_BUFFER));
if( result == 0 )
{
    if( out_par.ReturnStatus == 0 )
    {
        printf("\nWrite OK (%d)\n",
              );
    }
    else
    {
        printf("\nWrite failed (%d)\n",
              out_par.ReturnStatus);
    }
}
else
{
    printf( "\nSend failed --> Error = %d.\n", errno );
}

...
```

RETURNS

ReturnStatus	value	description
	0x00000000	OK
<i>TPMC821_ERR_PARAERR</i>	0x08210004	Illegal parameter value found
<i>TPMC821_ERR_DEVERR</i>	0x08210005	Hardware error or module not initialized
<i>TPMC821_ERR_TIMEOUT</i>	0x08210008	The command or an access timed out
<i>TPMC821_ERR_BUSSTOPPED</i>	0x0821000A	The bus is stopped
<i>TPMC821_ERR_NOMEM</i>	0x0821000F	The driver can not allocate enough memory

4.3 TPMC821_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 *command* code and *timeout* must be set in *TPMC821_IN_BUFFER*. The *bcmd* selection of union will be used for input parameters. (See below)

The *ReturnStatus* will be returned in *TPMC821_OUT_BUFFER*. No other values will be returned.

```
typedef struct
{
    int                cmdBit;        /* Command bit (0..13)          */
    unsigned short     cmdParam;     /* Command parameter           */
} TPMC821_BCMD_STRUCT;
```

cmdBit

Specifies the command bit.

cmdParam

Specifies the parameter for the bit command.

More information about the command bits and the parameter values can be found in the *User Manuals for the INTERBUS Generation 4* which are parts of the *TPMC821 Engineering Manual*.

EXAMPLE

```
#include <sys/kernel.h>
#include "tpmc821.h"

...

pid_t    pid;
int      result;
TPMC821_IN_BUFFER  in_par; /* input parameter */
TPMC821_OUT_BUFFER out_par; /* output parameter */

/* Execute bit command 0 with parameter 0*/
in_par->u.bcnd.cmdBit = 0;
in_par->u.bcnd.cmdParam = 0;
in_par->ControlCode = TPMC821_BIT_CMD;
in_par->timeout = 2; /* timeout after 2 seconds */
result = Send(pid, &in_par, &out_par,
              sizeof(TPMC821_IN_BUFFER), sizeof(TPMC821_OUT_BUFFER));
if( result == 0 )
{
    if(out_par.ReturnStatus == 0 )
    {
        printf("OK\n");
    }
    else
    {
        printf("\nExecuting bit command failed (%d)\n",
              out_par->ReturnStatus);
    }
}
else
{
    printf( "\nSend failed --> Error = %d.\n", errno );
}

...
```

RETURNS

ReturnStatus	value	description
	0x00000000	OK
<i>TPMC821_ERR_DEVERR</i>	0x08210005	Hardware error or module not initialized
<i>TPMC821_ERR_TIMEOUT</i>	0x08210008	The command or an access timed out
<i>TPMC821_ERR_ILLBIT</i>	0x08210009	Illegal Bit specified

4.4 TPMC821_MBX_WAIT

This function executes a mailbox command on the TPMC821 and waits for completion and returns a result.

The *command* code and *timeout* must be set in *TPMC821_IN_BUFFER*. The *mbx* selection of union will be used for input parameters. (See below)

The *ReturnStatus* will be returned in *TPMC821_OUT_BUFFER*. The data returned from the TPMC821 will be placed in the union selection *mbx*. (See below)

The selection *mbx* of the unions in *TPMC821_IN_BUFFER*, and *TPMC821_OUT_BUFFER* reserve space for only one data element. The application has to allocate a buffer where the whole command and result buffer fits in.

```
typedef struct
{
    int                Size;                /* Command/Result size in words          */
    unsigned short    Buffer[1];           /* Pointer to parameter/result buffer     */
} TPMC821_MBX_STRUCT;
```

Size

Specifies the length of the command buffer used with *TPMC821_IN_BUFFER* and returns the size of the result buffer used with *TPMC821_OUT_BUFFER*.

Buffer

Buffer specifies the 1st element of the command or result buffer.

More information about mailbox commands and parameter values can be found in the *User Manuals for the INTERBUS Generation 4* which are parts of the *TPMC821 Engineering Manual*.

EXAMPLE

```

#include <sys/kernel.h>
#include "tpmc821.h"

#define MAX_BUFFER_SIZE 100

...

pid_t pid;
int result;
int size;

TPMC821_IN_BUFFER *in_par; /* pointer to input parameter */
TPMC821_OUT_BUFFER *out_par; /* pointer to output parameter */

in_par = (TPMC821_IN_BUFFER*)malloc(MAX_BUFFER_SIZE);
out_par = (TPMC821_OUT_BUFFER*)malloc(MAX_BUFFER_SIZE);

/* asynchronous operating mode.*/
in_par->u.mbx.Size = 3;
in_par->u.mbx.Buffer[0] = 0x0702; /* Stop data transfer */
in_par->u.mbx.Buffer[1] = 1;
in_par->u.mbx.Buffer[2] = 0;

size = BUFIN_SIZE_RAW +
        sizeof(TPMC821_MBX_STRUCT) +
        (sizeof(unsigned short) * (in_par->u.mbx.Size - 1));

in_par->ControlCode = TPMC821_MBX_WAIT;
in_par->timeout = 2; /* timeout after 2 seconds */
result = Send(pid, in_par, out_par, size, MAX_BUFFER_SIZE);

if( result == 0 )
{
    if(out_par.ReturnStatus == 0 )
    {
        printParameter(&out_par->u.mbx.Buffer[0], out_par->u.mbx.Size);
        printf("OK\n");
    }
    else
    {
        printf("\nExecuting mailbox command failed (%d)\n",
            out_par->ReturnStatus);
    }
}
else

...

```

```

...

{
    printf( "\nSend failed --> Error = %d.\n", errno );
}

...

/* Print returned paramters */
void printParameter
(
    unsigned short    *pPar,
    int               num
)
{
    int               i;

    printf("\nPARAMETER:\n");
    for(i = 0; i < num; i++)
    {
        printf("%04X ", *(pPar++));
    }
    printf("\n");
}

...

```

RETURNS

ReturnStatus	value	description
	0x00000000	OK
<i>TPMC821_ERR_DEVERR</i>	0x08210005	Hardware error or module not initialized
<i>TPMC821_ERR_ILLBUFSIZE</i>	0x08210007	Specified buffer size is invalid
<i>TPMC821_ERR_TIMEOUT</i>	0x08210008	The command or an access timed out

4.5 TPMC821_MBX_NOWAIT

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

The *command* code and *timeout* must be set in *TPMC821_IN_BUFFER*. The *mbx* selection of union will be used for input parameters. (See below)

The *ReturnStatus* will be returned in *TPMC821_OUT_BUFFER*. No other values will be returned.

The selection *mbx* of the union in *TPMC821_IN_BUFFER* reserves space for only one data element. The application has to allocate a buffer where the whole command buffer fits in.

```
typedef struct
{
    int                Size;                /* Command/Result size in words          */
    unsigned short    Buffer[1];           /* Pointer to parameter/result buffer     */
} TPMC821_MBX_STRUCT;
```

Size

Specifies the length of the command buffer.

Buffer

Buffer specifies the 1st element of the command buffer.

More information about mailbox commands and parameter values can be found in the *User Manuals for the INTERBUS Generation 4* which are parts of the *TPMC821 Engineering Manual*.

EXAMPLE

```
#include <sys/kernel.h>
#include "tpmc821.h"

#define    MAX_BUFFER_SIZE    100

...

pid_t    pid;
int      result;
int      size;

TPMC821_IN_BUFFER    *in_par; /* pointer to input parameter */
TPMC821_OUT_BUFFER    out_par; /* output parameter */

in_par = (TPMC821_IN_BUFFER*)malloc(MAX_BUFFER_SIZE);

...

```

```

...

/* asynchronous operating mode. (not waiting for result) */
in_par->u.mbx.Size = 3;
in_par->u.mbx.Buffer[0] = 0x0702; /* Stop data transfer */
in_par->u.mbx.Buffer[1] = 1;
in_par->u.mbx.Buffer[2] = 0;

size = BUFIN_SIZE_RAW +
      sizeof(TPMC821_MBX_STRUCT) +
      (sizeof(unsigned short) * (in_par->u.mbx.Size - 1));

in_par->ControlCode = TPMC821_MBX_NOWAIT;
in_par->timeout = 2; /* timeout after 2 seconds */
result = Send(pid, in_par, &out_par, size, sizeof(TPMC821_OUT_BUFFER));

if( result == 0 )
{
    if(out_par.ReturnStatus == 0 )
    {
        printf("OK\n");
    }
    else
    {
        printf("\nExecuting mailbox command failed (%d)\n",
              out_par->ReturnStatus);
    }
}
else
{
    printf( "\nSend failed --> Error = %d.\n", errno );
}

...

```

RETURNS

ReturnStatus	value	description
	0x00000000	OK
<i>TPMC821_ERR_DEVERR</i>	0x08210005	Hardware error or module not initialized
<i>TPMC821_ERR_ILLBUFSIZE</i>	0x08210007	Specified buffer size is invalid
<i>TPMC821_ERR_TIMEOUT</i>	0x08210008	The command or an access timed out

4.6 TPMC821_GET_DIAG

This function executes reads diagnostic information.

The *command* code and *timeout* must be set in *TPMC821_IN_BUFFER*. No other values are used for input.

The *ReturnStatus* will be returned in *TPMC821_OUT_BUFFER*. The *diag* selection of union will be used for input parameters. (See below)

```
typedef struct
{
    unsigned short    sysfailReg;    /* contents of sysfail register    */
    unsigned short    configReg;     /* contents of config register     */
    unsigned short    diagReg;       /* contents of diagn. register     */
    unsigned char     hardwareFail;  /* HW failure has been detected    */
    unsigned char     initComplete;  /* HW init has completed with success */
} TPMC821_DIAG_STRUCT;
```

sysfailReg

Returns the actual value of the sysfailReg hardware register. (See *User Manuals for the INTERBUS Generation 4* which are parts of the *TPMC821 Engineering Manual*.)

configReg

Returns the actual value of the configReg hardware register. (See *User Manuals for the INTERBUS Generation 4* which are parts of the *TPMC821 Engineering Manual*.)

diagReg

Returns the actual value of the diagReg hardware register. (See *User Manuals for the INTERBUS Generation 4* which are parts of the *TPMC821 Engineering Manual*.)

hardwareFail

Returns *TRUE* if a hardware error has been detected, otherwise the value will be *FALSE*.

initComplete

Returns *TRUE* if the firmware has completed initialization, otherwise the value will be *FALSE*.

EXAMPLE

```
#include <sys/kernel.h>
#include "tpmc821.h"

...

pid_t    pid;
int      result;

TPMC821_IN_BUFFER  in_par; /* input parameter */
TPMC821_OUT_BUFFER out_par; /* output parameter */

in_par->ControlCode = TPMC821_GET_DIAG;
in_par->timeout = 2; /* timeout after 2 seconds */
result = Send(pid, &in_par, &out_par,
              sizeof(TPMC821_IN_BUFFER), sizeof(TPMC821_OUT_BUFFER));

if( result == 0 )
{
    if(out_par.ReturnStatus == 0 )
    {
        printf("OK\n");
    }
    else
    {
        printf("\nGet diagnostic failed (%d)\n",
              out_par->ReturnStatus);
    }
}
else
{
    printf( "\nSend failed --> Error = %d.\n", errno );
}

...
```

RETURNS

ReturnStatus	value	description
	0x00000000	OK

4.7 TPMC821_CONFIGURE

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

The *command* code and *timeout* must be set in *TPMC821_IN_BUFFER*. The *config* selection of union will be used for input parameters. (See below)

The *ReturnStatus* will be returned in *TPMC821_OUT_BUFFER*. No other values will be returned.

```
typedef struct
{
    unsigned long    op_mode;    /* operation mode                */
    long            dt_timeout;  /* Data Timeout in seconds       */
    long            mb_timeout;  /* Mailbox Timeout in seconds    */
} TPMC821_CONFIG_STRUCT;
```

op_mode

Specifies the new operation mode. For more information on operation modes see *User Manuals for the INTERBUS Generation 4* which are parts of the *TPMC821 Engineering Manual*. The following allowed values are defined in *tpmc821.h*:

Value	Meaning
<i>TPMC821_ASYNC</i>	asynchronous operation mode (default)
<i>TPMC821_ASYNC_LOCK</i>	asynchronous operation mode with consistency locking
<i>TPMC821_BUSSYNC</i>	bus synchronous mode
<i>TPMC821_PRGSYNC</i>	program synchronous

dt_timeout

Specifies the new timeout for data accesses. The value is specified in seconds.

mb_timeout

Specifies the new timeout for mailbox operations. The value is specified in seconds.

EXAMPLE

```

#include <sys/kernel.h>
#include "tpmc821.h"

...

pid_t    pid;
int      result;
TPMC821_IN_BUFFER  in_par;  /* input parameter */
TPMC821_OUT_BUFFER out_par; /* output parameter */

in_par->u.config.op_mode = TPMC821_ASYNC; /* asynchronous mode */
in_par->u.config.dt_timeout = 5;          /* data timeout: 5 sec */
in_par->u.config.mb_timeout = 8;         /* mailbox timeout: 8 sec */
in_par->ControlCode = TPMC821_CONFIGURE;
in_par->timeout = 2; /* timeout after 2 seconds */
result = Send(pid, &in_par, &out_par,
              sizeof(TPMC821_IN_BUFFER), sizeof(TPMC821_OUT_BUFFER));
if( result == 0 )
{
    if(out_par.ReturnStatus == 0 )
    {
        printf("OK\n");
    }
    else
    {
        printf("\nConfiguration failed (%d)\n",
              out_par->ReturnStatus);
    }
}
else
{
    printf( "\nSend failed --> Error = %d.\n", errno );
}

...

```

RETURNS

ReturnStatus	value	description
	0x00000000	OK
<i>TPMC821_ERR_PARAERR</i>	0x08210004	Illegal parameter value specified

4.8 TPMC821_SET_HOST_FAIL

This function sets the host fail 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 Engineering Manual*.

The *command* code and *timeout* must be set in *TPMC821_IN_BUFFER*. No other values will be used.

The *ReturnStatus* will be returned in *TPMC821_OUT_BUFFER*. No other values will be returned.

EXAMPLE

```
#include <sys/kernel.h>
#include "tpmc821.h"

...

pid_t    pid;
int      result;
TPMC821_IN_BUFFER in_par; /* input parameter */
TPMC821_OUT_BUFFER out_par; /* output parameter */

in_par->ControlCode = TPMC821_SET_HOST_FAIL;
in_par->timeout = 2; /* timeout after 2 seconds */
result = Send(pid, &in_par, &out_par,
              sizeof(TPMC821_IN_BUFFER), sizeof(TPMC821_OUT_BUFFER));
if( result == 0 )
{
    if(out_par.ReturnStatus == 0 )
    {
        printf("OK\n");
    }
    else
    {
        printf("\nSetting failed (%d)\n",
              out_par->ReturnStatus);
    }
}
else
{
    printf( "\nSend failed --> Error = %d.\n", errno );
}

...
```

RETURNS

ReturnStatus	value	description
	0x00000000	OK

4.9 TPMC821_REMOVE_HOST_FAIL

This function removes the host interrupt request that 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 are parts of the *TPMC821 Engineering Manual*.

The *command* code and *timeout* must be set in *TPMC821_IN_BUFFER*. No other values will be used.

The *ReturnStatus* will be returned in *TPMC821_OUT_BUFFER*. No other values will be returned.

EXAMPLE

```
#include <sys/kernel.h>
#include "tpmc821.h"

...

pid_t    pid;
int      result;
TPMC821_IN_BUFFER  in_par;  /* input parameter */
TPMC821_OUT_BUFFER out_par; /* output parameter */

in_par->ControlCode = TPMC821_REMOVE_HOST_FAIL;
in_par->timeout = 2; /* timeout after 2 seconds */
result = Send(pid, &in_par, &out_par,
              sizeof(TPMC821_IN_BUFFER), sizeof(TPMC821_OUT_BUFFER));
if( result == 0 )
{
    if(out_par.ReturnStatus == 0 )
    {
        printf("OK\n");
    }
    else
    {
        printf("\nRemoving failed (%d)\n",
              out_par->ReturnStatus);
    }
}
else
{
    printf( "\nSend failed --> Error = %d.\n", errno );
}

...
```

RETURNS

ReturnStatus	value	description
	0x00000000	OK

4.10TPMC821_CLEAR_HWERROR

This function clears the hardware error flag, which is set on service interrupt requests, which are 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 are parts of the *TPMC821 Engineering Manual*.

The *command* code and *timeout* must be set in *TPMC821_IN_BUFFER*. No other values will be used.

The *ReturnStatus* will be returned in *TPMC821_OUT_BUFFER*. No other values will be returned.

EXAMPLE

```
#include <sys/kernel.h>
#include "tpmc821.h"

...

pid_t    pid;
int      result;
TPMC821_IN_BUFFER  in_par; /* input parameter */
TPMC821_OUT_BUFFER out_par; /* output parameter */

in_par->ControlCode = TPMC821_CLEAR_HWERROR;
in_par->timeout = 2; /* timeout after 2 seconds */
result = Send(pid, &in_par, &out_par,
              sizeof(TPMC821_IN_BUFFER),sizeof(TPMC821_OUT_BUFFER));
if( result == 0 )
{
    if(out_par.ReturnStatus == 0 )
    {
        printf("OK\n");
    }
    else
    {
        printf("\nClear failed (%d)\n",
              out_par->ReturnStatus);
    }
}
else
{
    printf( "\nSend failed --> Error = %d.\n", errno );
}

...
```

RETURNS

ReturnStatus	value	description
	0x00000000	OK