
TPMC813-SW-42

VxWorks Device Driver

LON Interface

Version 1.0.x

User Manual

Issue 1.0

July 2004

TPMC813-SW-42

LON Interface

VxWorks 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.

©2004 by TEWS TECHNOLOGIES GmbH

Issue	Description	Date
1.0	First Issue	July 7, 2004

Table of Content

1	INTRODUCTION	4
2	INSTALLATION	5
	2.1 Install the driver to the VxWorks-System	5
	2.2 Including the driver in VxWorks	5
	2.3 Special installation for Intel x86 based targets.....	6
3	I/O SYSTEM FUNCTIONS	7
	3.1 tp813Drv()	7
	3.2 tp813DevCreate().....	8
4	I/O INTERFACE FUNCTIONS	10
	4.1 open()	10
	4.2 close().....	12
	4.3 read()	13
	4.4 write()	16
	4.5 ioctl()	20
5	APPENDIX	22
	5.1 Predefined Symbols.....	22
	5.2 Status and Error Codes	22

1 Introduction

The TPMC813-SW-42 VxWorks device driver allows the operation of the TPMC813 LON interface PMC conforming to the VxWorks I/O system specification. This includes a device-independent basic I/O interface with *open()*, *close()*, *read()*, *write()*, and *ioctl()*.

After installation of the device driver in the I/O system and creation of a device messages can be transmitted to and received from the LON bus by calling the *write()* or *read()* 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 queuing mechanism to prevent simultaneous requests by multiple users from interfering with each other.

To prevent the application for losing data each TPMC813 device contains a message FIFO of 30 messages.

This device driver supports explicit addressing only.

Attention, there may be problems with the structure alignment. The driver is tested with the GNU Toolkit and the Tornado Tools for VxWorks.

The used alignment allows packed bit fields, and byte alignment for all data types except structures and unions.

2 Installation

The software is delivered on a 3½" HD diskette.

Following files are located on the diskette:

tpmc813drv.c	TPMC813 driver source
tpmc813.h	TPMC813 include file for driver and application
tp813DEFS.H	TPMC813 driver include file
ni_msg.h	LONWORKS network interface protocol definitions
ni_mgmt.h	LONWORKS network management definitions
ni_lib.c	Interface for LONWORKS structures
tp813pci.c	PCI MMU mapping for Intel x86 based targets
tpxxxhwdep.c	Carrier specific adaptation functions
tpxxxhwdep.h	Carrier specific adaptation header
tp813tst.c	Test and example application

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

2.1 Install the driver to the VxWorks-System

You have to perform the following steps to install the TPMC813 device driver to the VxWorks target system.

- Build the object code of the TPMC813 device driver.
- Link or load the driver object file to your VxWorks-System.
- Call the 'tp813Drv()' function to install the driver.

2.2 Including the driver in VxWorks

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

2.3 Special installation for Intel x86 based targets

The TPMC813 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 is 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, we can't access the required PCI memory spaces.

To solve this problem we have to add a MMU mapping entry for the required TPMC813 PCI memory spaces prior the MMU initialization (*usrMmulnit()*) 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 we won't include the function *tp813PciInit()* and can skip the following steps.

The C source file **tp813pci.c** contains the function *tp813PciInit()*. This routine find out all TPMC813 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 (*usrMmulnit()*).

The right place to call the function *tp813PciInit()* is at the end of the function *sysHwInit()* in **sysLib.c** (can be opened from the project *Files* window).

Be sure that the function is called prior to MMU initialization otherwise the TPMC813 PCI spaces remains unmapped and we got an access fault during driver initialization.

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

```
tp813PciInit();
```

To link the driver object modules to VxWorks, simply add all necessary driver files to the project.

The Function *tp813PciInit()* was designed for and tested on generic Pentium targets. If you use another BSP please refer to BSP documentation or contact the technical support for required adaptation.

If you got strange errors after system startup with the new build system please carry 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 tp813Drv()

NAME

tp813Drv() - installs the TPMC813 driver in the I/O system.

SYNOPSIS

STATUS tp813Drv (void)

DESCRIPTION

This function installs the TPMC813 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

tpmc813.h

3.2 tp813DevCreate()

NAME

tp813DevCreate() - adds a TPMC813 device to the system and initialize device hardware.

SYNOPSIS

```
STATUS tp813DevCreate
(
    char      *Name,          /* name of the device to create      */
    int       modCount       /* number of TPMC813 that should be  */
)

```

DESCRIPTION

This routine is called to add a device to the system that will be serviced by the TPMC813 device driver. This function must be called before performing any I/O request to this device.

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

PARAMETER

Name

This argument specifies the name of the device to create. (i.e. "/tp813A")

modCount

This argument specifies the number of the TPMC813 the device should be created for. The first found module will be used when a 0 is specified, the second will be selected when a 1 is specified.

EXAMPLE

```
#include "tpmc813.h"

STATUS    status;

...

/*
-----
    Create a device "/tp813A" for TPMC813
    (crate the device on the second found TPMC813)
-----
*/
status = tpmc813DevCreate ("/tp813A", 1);

...
```

RETURNS

OK or ERROR

INCLUDE FILES

tpmc813.h

4 I/O interface functions

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

4.1 open()

NAME

open() - opens a device or file

SYNOPSIS

```
int open
(
    char    *name,          /* name of the device to open          */
    int     flags,         /* not used for TPMC813 driver, must be 0 */
    int     mode            /* not used for TPMC813 driver, must be 0 */
)
```

DESCRIPTION

Before I/O can be performed to the TPMC813 device, a file descriptor must be opened by invoking the basic I/O function *open()*.

EXAMPLE

```
int tp813dev;

...

/*
-----
  Open the device named "/tp813A" for I/O
-----
*/
tp813dev = open ("/tp813A", 0, 0);

...

```

RETURNS

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

SEE ALSO

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

4.2 close()

NAME

close() - closes a device or file

SYNOPSIS

```
int close
(
    int      dev_desc      /* device descriptor of the device */
)
```

DESCRIPTION

After closing the device it is no longer accessible for the application.

PARAMETER

dev_desc

This parameter must be the value returned from the open function for the device.

EXAMPLE

```
int  tp813dev;

...

/*
-----
  Close the device
-----
*/
close (tp813dev);

...
```

RETURNS

OK, if closing was successful executed.

SEE ALSO

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

4.3 read()

NAME

read() – reads a message from specified TPMC813 device

SYNOPSIS

```
int read
(
    int          fd,          /* device descriptor from opened TPMC813 device */
    tp813_RW *buffer,        /* pointer to message buffer */
    size_t       maxbytes    /* not used */
)
```

DESCRIPTION

Once a TPMC813 device has been opened, tasks can read uplink messages, responses and commands from the ECHELON controller over the MIP/DPS-based Network Interface (see also LONBUILDER Microprocessor Interface Program (MIP) User's Guide and LONWORKS Host Application Programmer's Guide).

If necessary a flush (set I/O flag *FLUSH*) of the device driver and LON controller message FIFO's will be performed before initiating the read request.

If the device is blocked by another read request or no message is available in the read buffer, the requesting task will be blocked until a message was read or the request times out.

PARAMETER

fd

This argument specifies the device descriptor to select the TIP813 device.

buffer

This argument points to a data structure (*tp813_RW*) holding the message information. The data structure is described below.

maxbytes

This argument is unused.

Data structure *tp813_RW*:

```
typedef struct
{
    unsigned long    io_flags,    /* I/O flags [FLUSH] */
    unsigned long    timeout,    /* The function call shall time out after n milliseconds */
    NetMsg           *buffer     /* I/O data buffer */
} tp813_RW;
```

io_flags

This argument specifies if a flush shall be executed before reading the data.

timeout

This argument specifies the maximum time the request will wait.

buffer

This pointer points to the data structure.

EXAMPLE

```
#include "tpmc813.h"
#include "ni_msg.h"
#include "ni_mgmt.h"

int            tp813dev;    /* device descriptor for open device */
unsigned long  result;
tp813_RW      rw;         /* I/O parameter block for de_read */
ExpAppBuffer  msg_in;

...

rw.flags      = 0;
rw.buffer     = &msg_in;
rw.timeout    = 1000;     /* 1000ms = 1 second*/
```

```
/*
-----
  Read a message from TPMC813 device.
  If no message available, the read request times out after
  1 second
-----
*/
result = read(tp813dev, &rw, 0);
if (result <= 0)
{
    /* handle the read error */
}
else
{
    /* process message */
}

...
```

RETURNS

If the return value is greater than zero the read access was successful, otherwise a read error occurred. If an error is occurred, the error number can be read from *errno*.

INCLUDE FILES

tpmc813.h, ni_msg.h, ni_mgmt.h

SEE ALSO

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

LONBUILDER Microprocessor Interface Program (MIP) User's Guide – Uplink Buffer Transfer and Local Command Processing

4.4 write()

NAME

write() – writes a message to a specified TPMC813 device

SYNOPSIS

```
int write
(
    int          fd,                /* device descriptor from opened TPMC813 device */
    tp813_RW *buffer,             /* pointer to message buffer */
    size_t       nbytes            /* not used because this information is part of the */
                                   /* tp813_RW structure */
)
```

DESCRIPTION

This function writes downlink messages and commands to the LON controller over the MIP/DPS-based Network Interface (see also LONBUILDER Microprocessor Interface Program (MIP) User's Guide and LONWORKS Host Application Programmer's Guide).

The TPMC813 device driver uses a global mechanism for returning status codes when errors occur. A global integer variable `errno` is set to an appropriate error code. The global variable `errno` is never cleared by the TPMC813 device driver. Thus, its value always indicates the last error status set.

PARAMETER

fd

This argument specifies the LON device.

buffer

This argument points to a data structure (`tp813_RW`) holding the message information (see below).

nbytes

This parameter is unused.

Data structure *tp813_RW*:

```
typedef struct
{
    unsigned long    io_flags,    /* I/O flags [FLUSH]                */
    unsigned long    timeout,     /* The function call shall time out after n milliseconds */
    NetMsg           *buffer      /* I/O data buffer                    */
} tp813_RW;
```

io_flags

This argument specifies the device descriptor to select the TIP813 device.

timeout

This message parameter specifies the amount of time in milliseconds that the task will wait for completion of the write request. If the device is blocked by another write request, the requesting task will be blocked until the transmitter becomes free or the request times out.

buffer

This pointer points to the data structure, where the data is delivered to the driver.

EXAMPLE

```
#include "tpmc813.h"
#include "ni_msg.h"
#include "ni_mgmt.h"

int            tp813dev;    /* device descriptor for open device */
unsigned long  result;
tp813_RW      rw;         /* I/O parameter block for write */
ExpAppBuffer  msg_out;
NM_set_node_mode_request *pdata;

...
```

```

/*
-----
    Preparing outgoing application buffer
    (Local Network Management Command using explicit addressing)
-----
*/
msg_out.ni_hdr.q.queue           = niTQ;
msg_out.ni_hdr.q.q_cmd           = niNETMGMT;
msg_out.ni_hdr.length           = sizeof(ExpMsgHdr) +
                                sizeof(ExplicitAddr) +
                                sizeof(NM_set_node_mode_request);

msg_out.msg_hdr.exp.tag          = 14;           /* magic cookie */
msg_out.msg_hdr.exp.auth        = 0;           /* not authenticated */
msg_out.msg_hdr.exp.st          = REQUEST;     /* service type */
msg_out.msg_hdr.exp.msg_type    = 0;           /* explicit message */
msg_out.msg_hdr.exp.response    = 0;           /* not a response msg */
msg_out.msg_hdr.exp.pool        = 0;           /* outgoing */
msg_out.msg_hdr.exp.alt_path    = 0;           /* use default path */
msg_out.msg_hdr.exp.addr_mode   = 0;           /* explicit addressing */
msg_out.msg_hdr.exp.cmpl_code   = MSG_NOT_COMPL;
msg_out.msg_hdr.exp.path        = 0;           /* use primary path */
msg_out.msg_hdr.exp.priority    = 0;           /* not a priority msg */
msg_out.msg_hdr.exp.length      = sizeof(NM_set_node_mode_request);
msg_out.addr.snd.ua.type        = A_LOCAL;

/*
-----
    Local Network Management Command (SET NODE MODE)
-----
*/
pdata          = (NM_set_node_mode_request*)&msg_out.data.exp;
pdata->code     = NM_set_node_mode;
pdata->mode     = CHANGE_STATE;
pdata->node_state = CNFG_ONLINE;

/*
-----
    Write the message to the TPMC813 device (see also example
    programs)
-----
*/
rw.pbuffer     = &msg_out;
result         = write(tp813dev, &rw, 0);

if (result <= 0)
{
    /* handle device error */
}

```

RETURNS

If the return value is greater than zero the read access was successful, otherwise a read error occurred. If an error is occurred, the error number can be taken from *errno*.

INCLUDE FILES

tpmc813.h, ni_msg.h, ni_mgmt.h

SEE ALSO

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

LONBUILDER Microprocessor Interface Program (MIP) User's Guide – Uplink Buffer Transfer and Local Command Processing

4.5 ioctl()

NAME

ioctl() - performs an I/O control function

SYNOPSIS

```
int ioctl
(
    int    fd,                /* device descriptor from opened TPMC813 device */
    int    function,         /* function code */
    int    arg                /* optional function dependent argument */
)
```

DESCRIPTION

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

PARAMETER

fd

This argument specifies the device descriptor to select the TIP813 device.

function

This parameter selects the action which will be executed by the driver. Following functions are supported:

Function code	description
FIO_RESET	This command performs a hardware reset of the LON controller. After a reset the LON controller enters a special <i>FLUSH</i> state. This state will be automatically cancelled from the device driver by sending the downlink command <i>niFLUSH_CANCEL</i>
FIO_FLUSH	This I/O control function flushes the read message FIFO of the device driver and all uplink application buffers of the LON controller.

arg

This argument is not used by the implemented functions.

EXAMPLE

```
#include "tpmc813.h"
#include "ni_msg.h"
#include "ni_mgmt.h"

int          tp813dev;      /* device descriptor for open device */
STATUS      result;

...

/*
-----
  Perform a hardware reset of the LON controller
-----
*/
result = ioctl(tp813dev, FIO_RESET, 0);

/*
-----
  Flush the read message FIFO and uplink application buffers
-----
*/
result = ioctl (tp813dev, FIO_FLUSH, 0);

...
```

RETURNS

OK or ERROR (if the device descriptor does not exist or the function code is unknown)

INCLUDE FILES

tpmc813.h, ni_msg.h, ni_mgmt.h

SEE ALSO

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

5 Appendix

This chapter describes the symbols which are defined in the file `tpmc813.h`.

5.1 Predefined Symbols

ioctl function codes

<code>FIO_FLUSH</code>	206	Command code for FIFO buffer flush
<code>FIO_RESET</code>	207	Command code to reset the LON controller

I/O flags

<code>TP813_FLUSH</code>	(1 << 3)	Flush read FIFO buffer
--------------------------	----------	------------------------

5.2 Status and 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()`.

<code>S_tp813Drv_NXIO</code>	0x08130100	No TPMC813 device is found at specified address
<code>S_tp813Dev_ICMD</code>	0x08130102	Illegal <code>ioctl</code> command
<code>S_tp813Dev_DEVBUSY</code>	0x08130003	Specified device is busy
<code>S_tp813Dev_TIMEOUT</code>	0x08130105	Request is timed out
<code>S_tp813Dev_BUFSIZE</code>	0x08130107	Illegal buffer length
<code>S_tp813Dev_IO</code>	0x0813010b	Device I/O error
<code>S_tp813Dev_BUSY</code>	0x0813010d	Specified device is busy