The Embedded I/O Company

TEWS TECHNOLOGIES

TPMC815-SW-72

LynxOS Device Driver

ARCNET Controller

Version 1.0.x

User Manual

Issue 1.0

May 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

TPMC815-SW-72

ARCNET Controller

LynxOS Device Driver

Issue	Description	Date
1.0	First Issue	May 5, 2004

Table of Content

1 Introduction

The TPMC815-SW-72 LynxOS device driver allows the operation of a TPMC815 ARCNET controller PMC on PowerPC platforms with DRM based PCI interface.

The standard file (I/O) functions (open, close, read, write, ioctl) provide the basic interface for opening and closing a file descriptor and for performing device I/O and configuration operations.

The TPMC815 device driver includes the following functions:

- Receiving and Sending messages
- Setup ARCNET controller and getting online
- Getting Offline
- Flushing receive buffer
- Getting diagnostic value

To understand all features of this device driver, it is recommended to read the TPMC815 User Manual.

2 Installation

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

The directory A:\TPMC815-SW-72 contains the following files:

TPMC815-SW-72.pdf	This manual in PDF format
TPMC815-SW-72.tar	Device Driver and Example sources

The TAR archive TPMC815-SW-72.tar contains the following files and directories:

tpmc815.c	Driver source code
tpmc815.h	Definitions and data structures for driver and application
tpmc815def.h	Definitions and data structures for the driver
arcnet.h	Definitions of controller values
tpmc815_info.c	Device information definition
tpmc815_info.h	Device information definition header
tpmc815.cfg	Driver configuration file include
tpmc815.import	Linker import file
Makefile	Device driver make file
example/example.c	Example application source

In order to perform a driver installation first extract the TAR file to a temporary directory then copy the following files to their target directories:

1. Create a new directory in the system drivers directory path /sys/drivers.xxx, where xxx represents the BSP that supports the target hardware.

   For example: /sys/drivers.pp_drm/tpmc815 or /sys/drivers.cpci_x86/tpmc815

2. Copy the following files to this directory:
   - tpmc815.c
   - tpmc815def.h
   - tpmc815.import
   - Makefile

3. Copy tpmc815.h to /usr/include/

4. Copy tpmc815_info.c to /sys/devices.xxx/ or /sys/devices if /sys/devices.xxx does not exist (xxx represents the BSP).

5. Copy tpmc815_info.h to /sys/dheaders/

6. Copy tpmc815.cfg to */sys/cfg.xxx/*, where xxx represents the BSP for the target platform

For example: /sys/cfg.ppc or /sys/cfg.x86 ....

2.1 Device Driver Installation

The two methods of driver installation are as follows:

- Static Installation
- Dynamic Installation (only native LynxOS systems)

2.1.1 Static Installation

With this method, the driver object code is linked with the kernel routines and is installed during system start-up.

2.1.1.1 Build the driver object

1. Change to the directory /sys/drivers.xxx/tpmc815, where xxx represents the BSP that supports the target hardware.
2. To update the library /sys/lib/libdrivers.a enter:

   ```
   make install
   ```

2.1.1.2 Create Device Information Declaration

1. Change to the directory /sys/devices.xxx/ or /sys/devices if /sys/devices.xxx does not exist (xxx represents the BSP).
2. Add the following dependencies to the Makefile

   ```
   DEVICE_FILES_all = ... tpmc815_info.x
   ```

   And at the end of the Makefile

   ```
   tpmc815_info.o:$(DHEADERS)/tpmc815_info.h
   ```

3. To update the library /sys/lib/libdevices.a enter:

   ```
   make install
   ```

2.1.1.3 Modify the Device and Driver Configuration File

In order to insert the driver object code into the kernel image, an appropriate entry in file CONFIG.TBL must be created.

1. Change to the directory /sys/lynx.os/ respective /sys/bsp.xxx, where xxx represents the BSP that supports the target hardware.
2. Create an entry at the end of the file CONFIG.TBL

   Insert the following entry at the end of this file. Be sure that the necessary TEWS TECHNOLOGIES IPAC carrier driver is included **before** this entry.

   ```
   I:tpmc815.cfg
   ```

2.1.1.4 Rebuild the Kernel

1. Change to the directory /sys/lynx.os/ (/sys/bsp.xxx)
2. Enter the following command to rebuild the kernel:

   ```
   make install
   ```

3. Reboot the newly created operating system by the following command (not necessary for KDIs):

   ```
   reboot –aN
   ```

   The N flag instructs init to run mknod and create all the nodes mentioned in the new nodetab.

4. After reboot you should find the following new devices (depends on the device configuration): /dev/tp815a, /dev/tp815b, /dev/tp815c, …

2.1.2 Dynamic Installation

This method allows you to install the driver after the operating system is booted. The driver object code is attached to the end of the kernel image and the operating system dynamically adds this driver to its internal structures. The driver can also be removed dynamically.

2.1.2.1 Build the driver object

1. Change to the directory /sys/drivers.xxx/tpmc815, where xxx represents the BSP that supports the target hardware.
2. To make the dynamic link-able driver enter :

   ```
   make dldd
   ```

2.1.2.2 Create Device Information Declaration

1. Change to the directory /sys/drivers.xxx/tpmc815, where xxx represents the BSP that supports the target hardware.
2. To create a device definition file for the major device (this works only on native system)

   ```
   make t815info
   ```

3. To install the driver enter:

   ```
   drinstall –c tpmc815.obj
   ```

   If successful, drinstall returns a unique <driver-ID>
4. To install the major device enter:

   ```
   devinstall –c –d <driver-ID> t815info
   ```

   The <driver-ID> is returned by the drinstall command
5. To create nodes for the devices enter:

   ```
   mknod /dev/tp815a c <major_no> 0
   ...
   ```

   The <major_no> is returned by the devinstall command.

If all steps are successful completed the TPMC815 is ready to use.

2.1.2.3 Uninstall dynamic loaded driver

To uninstall the TPMC815 device enter the following commands:

```
devinstall –u –c <device-ID>
drinstall –u <driver-ID>
```

2.1.3 Device Information Definition File

The device information definition contains information necessary to install the TPMC815 major device.

The implementation of the device information definition is done through a C structure, which is defined in the header file *tpmc815_info.h*.

This structure contains the following parameter:

PCIBusNumber Contains the PCI bus number at which the TPMC815 is connected. Valid bus numbers are in range from 0 to 255.

PCIDeviceNumber Contains the device number (slot) at which the TPMC815 is connected. Valid device numbers are in range from 0 to 31.

ModuleVariant Contains the module variant (-11/-21) for TPMC815 V1.0, ignored for V2.0.

If both PCIBusNumber and PCIDeviceNumber are –1 then the driver will auto scan for the TPMC815 device. The first device found in the scan order will be allocated by the driver for this major device.

Already allocated devices can't be allocated twice. This is important to know if there are more than one TMPC815 major devices.

A device information definition is unique for every TPMC815 major device. The file *tpmc815_info.c* on the distribution disk contains two device information declarations, **tp815a_info** for the first major device and **tp815b_info** for the second major device.

If the driver should support more than two major devices it is necessary to copy and paste an existing declaration and rename it with unique name for example **tp815c_info**, **tp815d_info** and so on.

It is also necessary to modify the device and driver configuration file respectively the configuration include file *tpmc815.cfg*.

The following device declaration information uses the auto find method to detect the TPMC815 module on PCI bus.

```
TP901_INFO tp815a_info = {

    -1,                 /*  Auto find the TPMC815 on any PCI bus   */
    -1,
    11,                 /*  module variant "11"                    */

};
```

2.1.4 Configuration File: CONFIG.TBL

The device and driver configuration file CONFIG.TBL contains entries for device drivers and its major and minor device declarations. Each time the system is rebuild, the config utility read this file and produces a new set of driver and device configuration tables and a corresponding nodetab.

To install the TPMC815 driver and devices into the LynxOS system, the configuration include file tpmc815.cfg must be included in the CONFIG.TBL (see also 2.1.1.3).

The file tpmc815.cfg on the distribution disk contains the driver entry (*C:tpmc815:*\....) and a major device entry (*D:TPMC815 1:t815a_info::*) with one minor device entry (*"N: tp815a"*).

If the driver should support more than one major device the following entries for major and minor devices must be enabled by removing the comment character (#). By copy and paste an existing major and minor entry and renaming the new entries, it is possible to add any number of additional TPMC815 devices.

This example shows a driver entry with one major device and one minor device:

```
#    Format:
#    C:driver-name:open:close:read:write:select:control:install:uninstall
#    D:device-name:info-block-name:raw-partner-name
#    N:node-name:minor-dev

C:tpmc815:\
     :tp815open:tp815close:tp815read:tp815write:\
     ::tp815ioctl:tp815install:tp815uninstall
D:TPMC815 1:tp815a_info::
N:tp815a:0
```

The configuration above creates the following node in the /dev directory.

/dev/tp815a

3 TPMC815 Device Driver Programming

LynxOS system calls are all available directly to any C program. They are implemented as ordinary function calls to "glue" routines in the system library, which trap to the OS code.

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.

3.1 open()

NAME

open() - open a file

SYNOPSIS

#include <sys/file.h>
#include <sys/types.h>
#include <fcntl.h>

int open (char *path, int oflags[, mode_t mode])

DESCRIPTION

Opens a file (TPMC815 device) named in ***path*** for reading and writing. The value of ***oflags*** indicates the intended use of the file. In case of a TPMC815 devices ***oflags*** must be set to **O_RDWR** to open the file for both reading and writing.

The ***mode*** argument is required only when a file is created. Because a TPMC815 device already exists this argument is ignored.

EXAMPLE

```
int fd

/* open the device named "/dev/tp815a" for I/O */
fd = open ("/dev/tp815a", O_RDWR);
```

RETURNS

open returns a file descriptor number if successful, or –1 on error.

SEE ALSO

LynxOS System Call - open()

3.2 close()

NAME

close() – close a file

SYNOPSIS

int close(int fd)

DESCRIPTION

This function closes an opened device.

EXAMPLE

```
int  result;

...

/*
**   close the device
*/
result = close(fd);

...
```

RETURNS

close returns 0 (OK) if successful, or –1 on error

SEE ALSO

LynxOS System Call - close()

3.3 read()

NAME

read() - read from a file

SYNOPSIS

#include <tpmc815.h>

int read (int fd, char *buff, int count)

DESCRIPTION

This function attempts to read a message from the TPMC815 associated with the file descriptor *fd* into a structure (*TP815_MSG*) pointed by *buff*. The argument *count* specifies the length of the buffer and must be set to the length of the structure *TP815_MSG*.

The *TP815_MSG* structure has the following layout:

```
typedef struct
{
        unsigned char       SID;
        unsigned char       DID;
        unsigned short      len;            /* Message data length                  */
        unsigned char       data[TP815_MAX_LONG_MSG];/* Message data                */
                                            /* up to 253 bytes data for SHORT PACKET */
                                            /* up to 508 bytes data for LONG PACKET  */
        unsigned long       io_flags;
        int                 timeout;
} TP815_MSG;
```

SID

Returns the source ID. The source ID is the ID of the node, which has send the message.

DID

Returns the target ID. This ID must be the ID of the actual target.

len

Returns the length of the message.

data[]

This array contains the received ARCNET message.

io_flags

This value is an ORed value of the following flags:

TP815_FLUSH	Flush receive buffer and wait for the next message receive.

timeout

Specifies the maximum time (in ticks) to wait for a message receive. If the time expires, the driver will return with a timeout error.

EXAMPLE

```
int           fd;
int           result;
TP815_MSG     readBuf;

/* Flush buffer and read message, timeout is 100 ticks */
readBuf.io_flags   = TP815_FLUSH;
readBuf.timeout    = 100;
result = read(fd, (char*)&readBuf, sizeof(readBuf));
if (result <= 0)
{
     // process error;
}
```

RETURNS

When *read* succeeds, the size of the read buffer (*TP815_MSG)* is returned. If read fails, -1 (SYSERR) is returned.

On error, *errno* will contain a standard read error code (see also LynxOS System Call – read) or a special error code (see also below Additional Error Codes)

SEE ALSO

LynxOS System Call - read()

TPMC815 example application

3.4 write()

NAME

write() – write to a file

SYNOPSIS

#include <tpmc815.h>

int write (int fd, char *buff, int count)

DESCRIPTION

This function attempts to write a message with the TP8MC15 associated with the file descriptor *fd* from a structure (TP815_MSG) pointed by buff. The argument count specifies the length of the buffer and must be set to the length of the structure TP815_MSG.

The *TP815_MSG* structure has the following layout:

```
typedef struct
{
        unsigned char       SID;
        unsigned char       DID;
        unsigned short      len;            /* Message data length                  */
        unsigned char       data[TP815_MAX_LONG_MSG];/* Message data                */
                                            /* - 253 bytes data for SHORT PACKET    */
                                            /* - 508 bytes data for LONG PACKET     */
        unsigned long       io_flags;
        int                 timeout;
} TP815_MSG;
```

SID

This parameter is unused.

DID

Specifies the target ID. The target ID selects the node the message should be sent to.

len

Specifies the length of the message

data[]

This array must contain the message that should be sent.

io_flags

This value is an ORed value of the following flags:

T815_NOWAIT	The function will return immediately and will not wait for successful transmission.

timeout

Specifies the maximum time (in ticks) to wait for a successful message send. If the time expires, the driver will return with a timeout error.

EXAMPLE

```
int                fd;
int                result;
TP815_MSG          sndBuf;

/* Send a message to node 5, timeout is 100 ticks */
sndBuf.DID         = 5;
sndBuf.timeout     = 100;
sndBuf.io_flags    = 0;
sndBuf.len         = strlen(“Hello”);
memcpy(sndBuf.data, “Hello”, sndBuf.len);

result = write(fd, (char*)&sndBuf, sizeof(sndBuf));
if (result <= 0)
{
     // process error;
}...
```

RETURNS

When *write* succeeds, the size of the write buffer (*TP815_MSG)* is returned. If write fails, -1 (SYSERR) is returned.

On error, *errno* will contain a standard read error code (see also LynxOS System Call – write) or a special error code (see also below Additional Error Codes)

SEE ALSO

LynxOS System Call - write()

TPMC815 example application

3.5 ioctl()

NAME

ioctl() – I/O device control

SYNOPSIS

#include <ioctl.h>
#include <tpmc815.h>

int ioctl (int fd, int request, char *arg)

DESCRIPTION

ioctl provides a way of sending special commands to a device driver. The call sends the value of request and the pointer arg to the device associated with the descriptor fd.

The following ioctl codes are supported by the driver and are defined in TPMC815.h:

Symbol	Meaning
TP815_C_ONLINE	Configure node and set node online
TP815_C_OFFLINE	remove node from net
TP815_C_DIAG	return diagnostic value
TP815_C_MAP	build a network map
TP815_C_FLUSH	flush receive buffer
TP815_C_TXSTATUS	return state of the last write request

See behind for more detailed information on each control code.

RETURNS

ioctl returns 0 if successful, or –1 on error.

On error, *errno* will contain a standard read error code (see also LynxOS System Call – ioctl) or a special error code (see also below Additional Error Codes)

SEE ALSO

LynxOS System Call - ioctl().

3.5.1 TP815_C_ONLINE

NAME

TP815_C_ONLINE – Configure node end set node online

DESCRIPTION

With this ioctl function the selected node will be setup and set online to the connected arcnet.

A pointer to the configuration structure (*TP815_CONFIG*) is passed by the parameter *arg* to the driver.

The *TP815_CONFIG* structure has the following layout:

```
typedef struct
{
        unsigned long           io_flags;
        unsigned char           node_id;            /* used if io_flag 'NODE_ID' is set          */
        unsigned char           network_timeout;    /* this parameter controlls the Response,    */
                                                    /* idle and Recon Times of our node in [us]  */
                                                    /* Value    Resp    Idle    Reconfig         */
                                                    /*   0      1130    1237     1680            */
                                                    /*   1       563     624     1680            */
                                                    /*   2       285     316     1680            */
                                                    /*   3        78      86      840            */
        unsigned char           speed;              /* this parameter determine the network speed*/
                                                    /*          Value          Network Speed     */
                                                    /*  TP815_5000KBPS      5       Mbps        */
                                                    /*  TP815_2500KBPS      2.5     Mbps        */
                                                    /*  TP815_1250KBPS      1.25    Mbps        */
                                                    /*  TP815_625KBPS       625     Kbps        */
                                                    /*  TP815_312KBPS       312.5   Kbps        */
} TP815_CONFIG;
```

io_flags

This parameter is an ORed value of following flags:

TP815_BROADCAST	Configure the controller to accept broadcast messages from the network.
TP815_LONGPACKET	Configure the controller to receive both short and long packets. If not set only short packets can be received.
TP815_NODE_ID	Use node ID specified in *TP815_CONFIG*. If not set, determine the node ID by reading the hardware DIP switch.

node_id

Specifies the node address on the network for this node. This node ID is only used if the I/O flag *TP815_NODE* is set. Valid values are 0 to 255.

network_timeout

Specifies the Response, Idle and Recon Times of the ARCNET controller. All nodes should be configured with the same timeout value for proper network operation. Valid values are shown in the following table:

For TPMC815-11 / -21:

Value	Response Time	Idle Time	Reconfig Time
0	596.8 µs	656 µs	840 ms
1	298.4 µs	328 µs	840 ms
2	74.7 µs	82 µs	840 ms
3	37.35 µs	41 µs	420 ms

speed

Determines the network speed by setting the clock prescaler to one of five possible values:

Name	Network Speed	Moduletypes
TP815_312KBPS	312.5 Kbps	-11 / -21
TP815_625KBPS	625 Kbps	-11 / -21
TP815_1250KBPS	1.25 Mbps	-11 / -21
TP815_2500KBPS	2.5 Mbps	-11 / -21
TP815_5000KBPS	5 Mbps	-11 / -21

EXAMPLE

```
int                fd;
int                result;
T815_CONF          cfgBuf;

/* Set node online (node: 5, 1.25 MBit, accept long packages,          */
/* Response Time: 37.35                                                */

cfgBuf.io_flags         = TP815_NODE_ID | TP815_LONGPACKET;
cfgBuf.node_id          = 5;
cfgBuf.speed            = TP815_1250KBPS;
cfgBuf.network_timeout  = 3;

result = ioctl(fd, TP815_C_ONLINE, (char*)&cfgBuf);
if (result != OK)
{
     // process error;
}
```

3.5.2 TP815_C_OFFLINE

NAME

TP815_C_OFFLINE – Removes the node from the net

DESCRIPTION

With this ioctl function the selected node will be removed from the net and set offline.

No parameter needed, the parameter *arg* must be set to *NULL*

EXAMPLE

```
int                fd;
int                result;

result = ioctl(fd, TP815_C_OFFLINE, NULL);
if (result != OK)
{
     // process error;
}
```

3.5.3 TP815_C_DIAG

NAME

TP815_C_DIAG – Read the number node reconfigurations

DESCRIPTION

This ioctl function will return the number of node reconfiguration occurred since the last read or start of initialization of the device.

A pointer to an unsigned int value is passed by the parameter *arg* to the driver, where the number of reconfigurations will be returned.

EXAMPLE

```
int                fd;
int                result;
unsigned int       numRecon;

result = ioctl(fd, TP815_C_DIAG, (char*)&numRecon);
if (result != OK)
{
     // process error;
}
```

3.5.4 TP815_C_MAP

NAME

TP815_C_MAP – Build a network map

DESCRIPTION

With this ioctl function a network map can be build, which identifies active nodes on the net.

A pointer to an array of 32 byte is passed by the parameter *arg* to the driver. Nodes which are set online will be marked with a set bit in the array. The node assignment look like the following.

The first byte of the array (index 0) returns the states of the nodes 0 to 7. Bit 0 represents node 0, node 1 can be checked with bit 1 and so on. The second byte of the array (index 1) returns the states of node 8 to 15 and so on.

Example:

map[0] = 0x02	node 1 is online, nodes 2 .. 7 are offline
map[1] = 0x03	nodes 8, 9 are online, nodes 10 .. 15 are offline
map[2] = 0x80	node 23 is online, nodes 16 .. 22 are offline
map[3] = 0x00	nodes 24 .. 31 are offline
...	...
map[31] = 0x00	nodes 248 .. 255 are offline

EXAMPLE

```
int                fd;
int                result;
unsigned char      map[32];

result = ioctl(fd, TP815_C_MAP, (char*)&map[0]);
if (result != OK)
{
     // process error;
}
```

3.5.5 TP815_C_FLUSH

NAME

TP815_C_FLUSH – Flush the receive buffer

DESCRIPTION

With this ioctl function the selected node will flush its receive buffer.

No parameter needed, the parameter *arg* must be set to *NULL*

EXAMPLE

```
int                fd;
int                result;

result = ioctl(fd, TP815_C_FLUSH, NULL);
if (result != OK)
{
     // process error;
}
```

3.5.6 TP815_C_TXSTATUS

NAME

TP815_C_TXSTATUS – Returns the state of the last write command

DESCRIPTION

With this ioctl function returns the state of the last write command. This command is useful for messages that are written with the *TP815_NOWAIT* option.

A pointer to an unsigned int value is passed by the parameter *arg* to the driver, where the status of the last write operation will be returned.

Possible values for txstatus are:

OK	transfer ok
EBUSY	transfer in progress
ENOTACKN	no acknowledge signal received
ETIMEDOUT	timeout during transfer

EXAMPLE

```
int                fd;
int                result;
unsigned int       txstatus;

result = ioctl(fd, TP815_C_TXSTATUS, (char*)&txstatus);
if (result != OK)
{
     // process error;
}
```

4 Debugging and Diagnostic

This driver was successfully tested on a Motorola MVME2300 board and on an Intel x86 native system. It was developed on a Windows Cross Environment for LynxOS V4.0.0.

If the driver will not work properly please enable debug outputs by defining the symbols *DEBUG*, *DEBUG_TPMC, DEBUG_PCI* and *DEBUG_INT.*

The debug output should appear on the console. If not please check the symbol *KKPF_PORT* in *uparam.h*. This symbol should be configured to a valid COM port (e.g. *SKDB_COM1*).

The debug output displays the device information data for the current major device, and a memory dump of the PCI base address registers like this.

```
Bus = 0  Dev = 16  Func = 0
[00] = 032F1498
[04] = 02800000
[08] = 02800000
[0C] = 00000008
[10] = 02042000
[14] = 0000C001
[18] = 0000D001
[1C] = 02043000
[20] = 00000000
[24] = 00000000
[28] = 00000000
[2C] = 000B1498
[30] = 00000000
[34] = 00000040
[38] = 00000000
[3C] = 00000109
PCI Base Address 0 (PCI_RESID_BAR0)

B8142000 : F1 FF FF 0F 00 FF FF 0F 00 00 00 00 00 00 00 00
B8142010 : 00 00 00 00 01 00 00 00 01 00 00 00 00 00 00 00
B8142020 : 00 00 00 00 00 00 00 00 E0 E0 01 54 E0 E0 01 54
B8142030 : 00 00 00 00 00 00 00 00 00 00 00 00 04 00 00 00
B8142040 : 0C 00 00 00 00 00 00 00 00 00 00 00 41 00 30 00
B8142050 : 00 00 78 18 6D 9B 24 02 00 00 00 00 00 00 00 00
B8142060 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B8142070 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
PCI Base Address 1 (PCI_RESID_BAR1)

B0108000 : F1 FF FF 0F 00 FF FF 0F 00 00 00 00 00 00 00 00
B0108010 : 00 00 00 00 01 00 00 00 01 00 00 00 00 00 00 00
B0108020 : 00 00 00 00 00 00 00 00 E0 E0 01 54 E0 E0 01 54
B0108030 : 00 00 00 00 00 00 00 00 00 00 00 00 04 00 00 00
B0108040 : 0C 00 00 00 00 00 00 00 00 00 00 00 41 00 30 00
B0108050 : 00 00 78 18 6D 9B 24 02 00 00 00 00 00 00 00 00
B0108060 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B0108070 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
PCI Base Address 2 (PCI_RESID_BAR2)

B0109000 : 03 00 45 00 80 00 18 00 01 01 01 01 01 01 01 01
B0109010 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
B0109020 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
B0109030 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
B0109040 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
B0109050 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
B0109060 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
B0109070 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
PCI Base Address 3 (PCI_RESID_BAR3)

B8143000 : F1 00 45 01 05 00 18 00 01 01 01 01 01 01 01 01
B8143010 : F1 00 45 02 00 00 18 00 01 01 01 01 01 01 01 01
B8143020 : F1 00 45 03 BA 00 18 00 01 01 01 01 01 01 01 01
B8143030 : F1 00 45 04 50 00 18 00 01 01 01 01 01 01 01 01
B8143040 : F1 00 45 05 2A 00 18 00 01 01 01 01 01 01 01 01
B8143050 : F1 00 45 06 6D 00 18 00 01 01 01 01 01 01 01 01
B8143060 : F1 00 45 07 A1 00 18 00 01 01 01 01 01 01 01 01
B8143070 : F1 00 45 08 6D 00 18 00 01 01 01 01 01 01 01 01
Found a TPMC815-11 V2.0 (HW-NodeID=0x01)  BusNo=0  DevNo=16
RegSpace=0xB0109000   MemSpace=0xB8143000
```

The debug output above is only an example. Debug output on other systems may be different for addresses and data in some locations.

5 Additional Error Codes

Error	Value	Description
ENOTREADY	801	node is not online
ENOTACKN	802	No ACK from receiver after transmission
EBADPACKETSIZE	803	illegal packet size
EBADSPEED	804	invalid bit-rate selected
EQFULL	805	The Queue for incoming messages is full
EDUPID	806	illegal (duplicate) node id