TEWS TECHNOLOGIES

TPMC500-SW-72
LynxOS Device Driver
TPMC500 – 32/16 12 Bit ADC

Version 1.0.x

Reference Manual

Issue 1.0

November 2001

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

TPMC500-SW-72

32/16 Channel 12-Bit ADC
LynxOS Device Driver

Issue	Description	Date
1.0	First Issue	November 7, 2001

Table of Contents

1 Introduction

The TPMC500-SW-72 LynxOS device driver allows the operation of a TPMC500 32(16) Channel 12 Bit ADC PMC on a PowerPC platform with DRM based PCI interface.

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

The TPMC500 device driver includes the following functions:

- ☞ read the ADC value from specified channel
- ☞ setup and start the ADC sequencer mode
- ☞ read data while sequencer mode is active (synchronous and wsynchronous)
- ☞ read module information
- ☞ all reads can be made with data correction. using the factory set data

2 Installation

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

Following files are located on the diskette:

tpmc500.c	Driver source code
tpmc500.h	Definitions and data structures for driver and application
tpmc500_info.c	Device information definition
tpmc500_info.h	Device information definition header
tpmc500.cfg	Driver configuration file include
tpmc500.import	Linker import file
Makefile	Device driver make file
Makefile.dldd	Make file for dynamic driver installation
tpmc500-sw-72.pdf	This Manual in PDF format

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.

In order to perform a static installation, copy the following files to the target directories:

1. Create a new directory in the system drivers directory path.
   For example: */sys/drivers.pp_drm/tpmc500*
2. Copy the following files to this directory: *tpmc500.c, Makefile*
3. Copy *tpmc500.h* to */usr/include/*
4. Copy *tpmc500_info.c* to */sys/devices/*
5. Copy *tpmc500_info.h* to */sys/dheaders/*
6. Copy *tpmc500.cfg* to */sys/cfg.ppc/*

2.1.1.1 Build the driver object

1. Change to the directory */sys/drivers.pp_drm/tpmc500*
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/*
2. Add the following dependencies to the *Makefile*

   ```
   DEVICE_FILES_prep = ...tpmc500_info.x
   ```

   And at the end of the Makefile

   ```
   ...
   tpmc500_info.o:$(DHEADERS)/tpmc500_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/
2. Create an entry in the file CONFIG.TBL
   Insert the entry after the console driver section

   ```
   # End of console devices
   I:tpmc500.cfg
   ```

2.1.1.4 Rebuild the Kernel

1. Change to the directory /sys/lynx.os/ (/sys/bsp.pp_drm)
2. To rebuild the kernel enter the following command:

   ```
   make install
   ```

3. Reboot the newly-created operating system by the following command:

   ```
   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/tp500a, [/dev/tp500b, …]*

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.
Unlike the description of the dynamic installation in the manual “Writing Device Drivers for LynxOS”, the driver source must be placed in a directory under */sys/drivers.pp_drm/*

The following steps describe how to do a dynamic installation:

1. Create a new directory in the system drivers directory path.
   For example: */sys/drivers.pp_drm/tpmc500*

2. Copy the following files to this directory:
   - tpmc500.c
   - tpmc500_info.c
   - tpmc500_info.h
   - tpmc500.import
   - Makefile.dldd

3. Copy *tpmc500.h* to */usr/include*

4. Change to the directory */sys/drivers.pp_drm/tpmc500*

5. To make the dynamic link-able driver enter :

   ```
   make –f Makefile.dldd
   ```

6. Create a device definition file for one major device

   ```
   gcc –DDLDD –o tpmc500_info tpmc500_info.c
   ./tpmc500_info > tp500a
   ```

7. To install the driver enter:

   ```
   drinstall –c tpmc500.obj
   ```

   If successful *drinstall* returns a unique <*driver-ID*>

8. To install the major device enter:

   ```
   devinstall –c –d <driver-ID> tp500a
   ```

   The <*driver-ID*> is returned by the *drinstall* command

9. To create nodes for the devices enter:

   ```
   mknod /dev/tp500a c <major_no> 0
   ...
   ```

If all steps are successful completed the TPMC500 is ready to use.

To uninstall the TPMC500 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 TPMC500 major device.
The implementation of the device information definition is done through a C structure which is defined in the header file *tpmc500_info.h*.

This structure contains following parameter:

PCIBusNumber	Contains the PCI bus number at which the TPMC500 is connected. Valid bus numbers are in range from 0 to 255.
PCIDeviceNumber	Contains the device number (slot) at which the TPMC500 is connected. Valid device numbers are in range from 0 to 31.

NOTE. If both *PCIBusNumber* and *PCIDeviceNumber* are –1 then the driver will auto scan for the TPMC500 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 you have more than one TMPC500 major device.

A device information definition is unique for every TPMC500 major device. The file *tpmc500_info.c* on the distribution disk contains two device information declarations, **tp500a_info** for the first major device and **tp500b_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 **tp500c_info**, **tp500d_info** and so on.

NOTE. It is also necessary to modify the device and driver configuration file respectively the configuration include file *tpmc500.cfg*.

The following device declaration information uses the auto find method to detect the TPMC500 module on PCI bus.

```
TP500_INFO tp500a_info = {

    -1,                 /*  auto find the TPMC500 on any PCI bus    */
    -1,

};
```

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 reads this file and produces a new set of driver and device configuration tables and a corresponding *nodetab*.

To install the TPMC500 driver and devices into the LynxOS system, the configuration include file *tpmc500.cfg* must be included in the *CONFIG.TBL* (see also 2.1.1.3).
The file *tpmc500.cfg* on the distribution disk contains the driver entry (C:tpmc500:\....) and one enabled major device entry (D:TPMC500 1:tp500a_info::) with one minor device entry (N: tp500a:0).

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 TPMC500 device.

NOTE. The name of the device information declaration (info-block-name) must match to an existing C structure in the file *tpmc500_info.c*.

This example shows a driver entry with one major device and 8 minor devices:

```
#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:tpmc500:\
 :tp500open:tp500close:tp500read::\
 ::tp500ioctl:tp500install:tp500uninstall
D:TPMC500 1:tp500a_info::
N:tp500a:0
```

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

```
/dev/tp500a
```

3 TPMC500 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 (TPMC500 device) named in ***path*** for reading and writing. The value of ***oflags*** indicates the intended use of the file. In case of a TPMC500 devices ***oflags*** must be set to **O_RDONLY** to open the file for reading.
The ***mode*** argument is required only when a file is created. Because a TPMC500 device already exists this argument is ignored.

EXAMPLE

```
int  fd

/*
**  open the device named "/dev/tp500a" for Input
*/

fd = open ("/dev/tp500a", O_RDONLY);
```

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 <tpmc500.h>

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

DESCRIPTION

The read function reads an ADC value from the specified channel.
The argument ***buff*** contains a pointer to the read buffer (TP500_READ_BUFFER), which contains information of the desired read operation.
The argument ***count*** is not required and should be 0.

The *TP500_READ_BUFFER* structure has the following layout:

```
typedef struct
{
    unsigned short      channel;        /* channel number */
    unsigned short      gain;           /* selected gain */
    unsigned short      flags;
    short               value;          /* ADC input value */
} TP500_READ_BUFFER, *PTP500_READ_BUFFER;
```

The parameter **channel** specifies the ADC channel that will be used. Allowed values are 1 to 32 for single-ended input and 1 to 16 for differential input.

The parameter **gain** specifies the input gain that will be used, following table shows the allowed values. These values are predefined in ‘tpmc500.h’.

Name	TPMC500-10/-12/-20/-22	TPMC500-11/-13/-21/-23
TP500_GAIN1	gain = 1	gain = 1
TP500_GAIN2	gain = 2	gain = 2
TP500_GAIN4	not supported	gain = 4
TP500_GAIN5	gain = 5	not supported
TP500_GAIN8	not supported	gain = 8
TP500_GAIN10	gain = 10	not supported

The parameter **flags** value is an ORed value of the flags shown in the following table.

Name	Meaning
TP500_FL_DIFF	If this flag is set, the driver will use differential input signal. If the flag is not set, the driver will use single-ended input signal.
TP500_FL_CORR	If this flag is set, the driver will correct the ADC input value with the factory programmed correction data. If this flag is not set, the driver will return the ADC input.
TP500_FL_FAST	If this flag is set, the driver will start a conversion on the last programmed channel, with the last selected gain and the last selected input mode. The parameters *gain*, *channel* and the flag *TP500_FL_DIFF* will be ignored if this flag is set. If this flag is used, the hardware coded settling time is not needed and not used, this makes the access faster. If the flag is not set, the driver will work in the normal mode.

The parameter **value** returns the converted ADC value.

EXAMPLE

```
int                    fd;
int                    result;
unsigned short         value;
TP500_READ_BUFFER      ReadBuf;

...

/*******************************************
 Read channel 5 with differential input
 use gain 2
 correct the input data
*******************************************/
ReadBuf.channel      = 5;
ReadBuf.gain         = TP500_GAIN2;
ReadBuf.flags        = TP500_FL_DIFF | TP500_FL_CORR;


result = read(fd, (char*)&ReadBuf, 0);


/*
**  Check the result of the last device I/O operation
*/

if( result == sizeof(TP500_READ_BUFFER)) {
  printf(“ADC value = 0x%x\n”, ReadBuf.value);
}
else {
  printf( "\nRead failed --> Error = %d.\n", errno );
}
```

RETURNS

When *read* succeeds, the size of the read buffer 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 one of the following TPMC500 specific error codes:

ENXIO	Invalid minor device specified.
EBUSY	Sequencer mode is active. This function can not be called while the sequencer mode is active.
ETIMEDOUT	The maximum allowed time to finish the read request is exhausted.
EINVAL	Invalid parameter. Please check the parameter.

SEE ALSO

LynxOS System Call - read()

3.4 ioctl()

NAME

ioctl() - I/O device control

SYNOPSIS

```
#include <ioctl.h>
#include <tpmc500.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 request values are support by a TPMC500 device :

Value	**Meaning**
TP500_READPARAM	Read module parameters, this includes the model type and the correction data
TP500_SEQSTOP	Stop sequencer, set module to normal mode
TP500_SEQSETUP	Setup and start sequencer, set module in sequencer mode
TP500_SEQREAD	Read sequencer data, synchronize with cycle time and get values
TP500_SEQIMMREAD	Read sequencer data, make an immediate read, retuning the last converted values

See behind for more detailed information on each control code.

Note

To use these TPMC500 specific control codes the header file
tpmc500.h must be included in the application.

RETURNS

ioctl returns 0 if successful, or –1 on error.
The TPMC500 ioctl function returns always standard error codes. See LynxOS system call ioctl of a detailed description of possible error codes.

SEE ALSO

LynxOS System Call - ioctl().

3.4.1 TP500_READPARAM - (Read Module Parameters)

The function *TP500_READPARAM* reads the module parameters of the TPMC500 including the model type and the factory programmed correction data.
The argument ***arg*** contains a pointer to the *TP500_PARA_BUFFER* data structure.

The *TP500_PARA_BUFFER* structure has the following layout:

```
typedef struct
{
    int              ModuleType;     /* TPMC500 variant type */
    signed char      OffsCorr[4];    /* Offset correction Data */
    signed char      GainCorr[4];    /* Gain correction Data */
} TP500_PARA_BUFFER, *PTP500_PARA_BUFFER;
```

The entry **ModelType** returns the model type. A value of 10 specifies a TPMC500-10, a value of 11 specifies a TPMC500-11 and so on.
The array **OffsCorr** returns the offset correction data. The index of the array specifies the gain the value is assigned to. (see table below)
The array **GainCorr** returns the gain correction data. The index of the array specifies the gain the value is assigned to. (see table below)

Index	**TPMC500-10/-12/-20/-22**	**TPMC500-11/-13/-21/-23**
0	Gain = 1	Gain = 1
1	Gain = 2	Gain = 2
2	Gain = 5	Gain = 4
3	Gain = 10	Gain = 8

Note

More information about data correction is printed in the
TPMC500 User Manual

EXAMPLE

```
int                  fd;
int                  result;
TP500_PARA_BUFFER    ParamBuf;

...

/*
**  Read module parameters
*/
result = ioctl (fd, TP500_READPARAM, (char*)&ParamBuf);

if (result < 0) {
  /* handle ioctl error */
}
```

3.4.2 TP500_SEQSTOP - (Stop Sequencer Mode)

The function *TP500_SEQSTOP* stops the sequencer and returns the module to normal mode.
The argument ***arg*** is unused and should be set to zero.

EXAMPLE

```
int                   fd;
int                   result;

...

/*
**  Get module parameters
*/
result = ioctl (fd, TP500_SEQSTOP, 0);

if (result < 0) {
  /* handle ioctl error */
}
```

3.4.3 TP500_SEQSETUP - (Setup and Start Sequencer Mode)

The function *TP500_SEQSETUP* sets up the TPMC550 to work in sequencer mode. The cycle time and the channel configuration are set up.
The argument ***arg*** contains a pointer to the TP500_SEQSET_BUFFER data structure.

The TP500_SEQSET_BUFFER structure has the following layout:

```
typedef struct
{
    unsigned short      cycleTime;        /* value of cycletime register */
    struct
    {
        unsigned short  flags;
        unsigned short  gain;             /* selected gain */
    } channel[TP500_SNGLCHANS];           /* channel configuration */
} TP500_SEQSET_BUFFER, *PTP500_SEQSET_BUFFER;
```

The entry **cycleTime** specifies the cycle time that will be used. The value will be copied into the sequencer timer register. The value has a resolution of 100µs steps. If this value is set to zero, the sequencer will work in continuous mode.

The structure **channel** holds information for the channels. The index of the channel structure specifies the channel. Index 0 is advised to channel 1, index 1 is advised to channel 2 and so on. The array has 32 elements. The structure contains the following entries.

The **flags** parameter is an ORed value of the following described flags.

Name	Meaning
TP500_FL_DIFF	If this flag is set, the driver will use differential input signal. If the flag is not set, the driver will use single-ended input signal.
TP500_FL_CORR	If this flag is set, the driver will correct the ADC input value with the factory programmed correction data. If this flag is not set, the driver will return the ADC input.
TP500_FL_ENABLE	If this flag is set the channel will be used in sequencer mode. If this flag is not set, the channel will be ignored in sequencer mode.

This **gain** parameter specifies the gain for the channel.

Name	TPMC500-10/-12/-20/-22	TPMC500-11/-13/-21/-23
TP500_GAIN1	gain = 1	gain = 1
TP500_GAIN2	gain = 2	gain = 2
TP500_GAIN4	not supported	gain = 4
TP500_GAIN5	gain = 5	not supported
TP500_GAIN8	not supported	gain = 8
TP500_GAIN10	gain = 10	not supported

EXAMPLE

```
int                   fd;
int                   result;
TP500_SEQSET_BUFFER   SeqSetBuf;

...

/**********************************************************
 Start sequencer with a cycle time of 1 sec
 Enable following channels:
    Channel 1: Gain=1, Correction enabled, single-ended
    Channel 6: Gain=2, Correction disabled, differential

SeqSetBuf.cycleTime = 10000;     /* 10000 * 100µs */

for (i = 0; i < TP500_SNGLCHANS; i++) {
    SeqSetBuf.channel[i].flags = 0;   /* disable channel */
}

SeqSetBuf.channel[0].flags = TP500_FL_ENABLE | TP500_FL_CORR;
SeqSetBuf.channel[5].flags = TP500_FL_ENABLE | TP500_FL_DIFF;

SeqSetBuf.channel[0].gain = TP500_GAIN1;
SeqSetBuf.channel[5].gain = TP500_GAIN2;

result = ioctl (fd, TP500_SEQSETUP, (char*)&SeqSetBuf);

if (result < 0) {
  /* handle ioctl error */
}
```

3.4.4 TP500_SEQREAD - (Read in Sequencer Mode)

The function *TP500_SEQREAD* returns ADC data in sequencer mode. This function returns if new data is available, it returns immediately, if unread data is present or it waits until the conversion cycle is completed.
The argument ***arg*** contains a pointer to the *TP500_SEQREAD_BUFFER* data structure.

The *TP500_SEQREAD_BUFFER* structure has the following layout:

```
typedef struct
{
    int             overrunCount;   /* number of lost cycles */
    int             error;          /* error flags */
    short           values[TP500_SNGLCHANS];
                                    /* ADC input value */
} TP500_SEQREAD_BUFFER, *PTP500_SEQREAD_BUFFER;
```

The parameter **overrunCount** returns the number of lost sequencer cycles. A value of '-1' means there has not been a valid cycle since last read (only in error case), a value of '0' means no data has been lost. If the value is greater '0', the value specifies the number of lost cycles.

The **error** value returns an ORed value of the following error flags. This value should be checked for every call of the function.

Name	Meaning
TP500_FL_HWOVERRUN	The hardware has detected an overflow, the data sequencer has not been serviced in one cycle time.
TP500_FL_TIMERERR	The hardware has signaled, that the specified cycle time is to short to make the specified conversions.
TP500_FL_INSTRAMERR	The hardware has detected an error in the instruction RAM. (No channel selected)
TP500_FL_SWOVERRUN	The driver can not make the data corrections in one cycle time.

The array **values** returns a full set of ADC values. Only the values of the channels selected in *TP500_SEQSETUP* will be valid. The index specifies the channel. Index 0 is advised to channel 1, index 1 is advised to channel 2 and so on. The array has 32 elements.

EXAMPLE

```
int                  fd;
int                  result;
TP500_SEQREAD_BUFFER SeqReadBuf;

...

result = ioctl (fd, TP500_SEQREAD, (char*)&SeqReadBuf);

if (result < 0) {
  /* handle ioctl error */
}
```

3.4.5 TP500_SEQIMMREAD - (Immediate Read in Sequencer Mode)

The function *TP500_SEQREAD* returns ADC data in sequencer mode. This function returns immediately and returns the last converted data also if it is not actualized since the last read.
The argument ***arg*** contains a pointer to the *TP500_SEQREAD_BUFFER* data structure.

The *TP500_SEQREAD_BUFFER* structure has the following layout:

```
typedef struct
{
   int            overrunCount;   /* number of lost cycles */
   int            error;          /* error flags */
   short          values[TP500_SNGLCHANS];
                                  /* ADC input value */
} TP500_SEQREAD_BUFFER, *PTP500_SEQREAD_BUFFER;
```

The parameter **overrunCount** returns the number of lost sequencer cycles. A value of '-1' means there has not been a valid cycle since the last read, the function will return the same data, a value of '0' means no data has been lost and a data update has occurred. If the value is greater '0', the value specifies the number of lost cycles.

The **error** value returns an ORed value of the following error flags. This value should be checked for every call of the function.

Name	Meaning
TP500_FL_HWOVERRUN	The hardware has detected an overflow, the data sequencer has not been serviced in one cycle time.
TP500_FL_TIMERERR	The hardware has signaled, that the specified cycle time is to short to make the specified conversions.
TP500_FL_INSTRAMERR	The hardware has detected an error in the instruction RAM. (No channel selected)
TP500_FL_SWOVERRUN	The driver can not make the data corrections in one cycle time.

The array **values** returns a full set of ADC values. Only the values of the channels selected in *TP500_SEQSETUP* will be valid. The index specifies the channel. Index 0 is advised to channel 1, index 1 is advised to channel 2 and so on. The array has 32 elements.

EXAMPLE

```
int                  fd;
int                  result;
TP500_SEQREAD_BUFFER SeqReadBuf;

...

result = ioctl (fd, TP500_SEQIMMREAD, (char*)&SeqReadBuf);

if (result < 0) {
  /* handle ioctl error */
}
```

4 Debugging

This driver was successful tested on a Motorola MVME3600-1 (PMCSPAN) and MVME2305-900 board in a native LynxOS environment and a Windows Cross development.

If the driver will not work properly, usually a PCI bus or interrupt problem, you can enable debug outputs by removing the comments around the symbols *DEBUG*, *DEBUG_PCI* and *DEBUG_TPMC*. The debug output will appear on the console.

The debug output displays the PCI Header, the address of each base address register and a memory dump of all mapped memory and I/O spaces of the TPMC500 like this (see also *TPMC500 User Manual – PCI Configuration*).

```
TPMC500 Device Driver Install
Bus = 0  Dev = 16  Func = 0
[00] = 905010B5
[04] = 02800000
[08] = 11800001
[0C] = 00000008
[10] = 02042000
[14] = 0000C001
[18] = 0000D001
[1C] = 02043000
[20] = 00000000
[24] = 00000000
[28] = 00000000
[2C] = 01F41498
[30] = 00000000
[34] = 00000000
[38] = 00000000
[3C] = 00000109
PCI Base Address 0 (PCI_RESID_BAR0)

E8142000 : 01 FF FF 0F 00 F8 FF 0F 00 00 00 00 00 00 00 00
E8142010 : 00 00 00 00 01 00 00 00 01 08 00 00 00 00 00 00
E8142020 : 00 00 00 00 00 00 00 00 42 A9 52 A9 C0 79 33 E9
E8142030 : 00 00 00 00 00 00 00 00 00 00 00 00 41 00 00 00
E8142040 : A1 00 00 00 E1 00 00 00 01 0C 00 00 49 00 00 00
E8142050 : 44 0B 78 18 00 00 00 00 00 00 00 00 00 00 00 00
E8142060 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
E8142070 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
PCI Base Address 1 (PCI_RESID_BAR1)

E0108000 : 01 FF FF 0F 00 F8 FF 0F 00 00 00 00 00 00 00 00
E0108010 : 00 00 00 00 01 00 00 00 01 08 00 00 00 00 00 00
E0108020 : 00 00 00 00 00 00 00 00 42 A9 52 A9 C0 79 33 E9
E0108030 : 00 00 00 00 00 00 00 00 00 00 00 00 41 00 00 00
E0108040 : A1 00 00 00 E1 00 00 00 01 0C 00 00 49 00 00 00
E0108050 : 44 0B 78 18 00 00 00 00 00 00 00 00 00 00 00 00
E0108060 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
E0108070 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
PCI Base Address 2 (PCI_RESID_BAR2)

E0109000 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
E0109010 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
E0109020 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
E0109030 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
E0109040 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
E0109050 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
E0109060 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
E0109070 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
PCI Base Address 3 (PCI_RESID_BAR3)

E8143000 : F8 0A F9 09 FA 02 F8 06 FF FF FF FF FF FF FF FF
E8143010 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
E8143020 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
E8143030 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
E8143040 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
E8143050 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
E8143060 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
E8143070 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF

Moduletype TPMC500-10
```