

TPMC917-SW-82

Linux Device Driver

4 MB SRAM with Battery Backup

and 4 Channel Serial Interface

Version 1.0.x

User Manual

Issue 1.0.0

May 2005

TPMC917-SW-82

4 MB SRAM with Battery Backup and
4 Channel Serial Interface

Linux 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.

©2005 by TEWS TECHNOLOGIES GmbH

Issue	Description	Date
1.0.0	First Issue	May 23, 2005

Table of Contents

1	INTRODUCTION.....	4
2	INSTALLATION.....	5
	2.1 Build and install the device driver.....	5
	2.2 Uninstall the device driver	6
	2.3 Install device driver into the running kernel	7
	2.4 Remove device driver from the running kernel	8
	2.5 Change Major Device Numbers	8
3	SERIAL DEVICE DRIVER PROGRAMMING	9
	3.1 Simple Programming example	9
	3.2 ioctl()	10
	3.2.1 TPMC917_IOCQ_BIST.....	11
4	SRAM DEVICE DRIVER PROGRAMMING.....	14
	4.1 Device Input/Output functions.....	14
	4.1.1 open().....	14
	4.1.2 close().....	16
	4.1.3 mmap().....	17
	4.1.4 munmap().....	19
	4.1.5 ioctl().....	21
	4.1.5.1 TP917RAM_IOCQ_GETMEMSIZE	23
	4.1.5.2 TP917RAM_IOCX_READ	24
	4.1.5.3 TP917RAM_IOCX_WRITE.....	26
	4.1.5.4 TP917RAM_IOCQ_GETBATSTAT	28
	4.1.5.5 TP917RAM_IOC_WAITBATLOW	29
5	DIAGNOSTIC.....	30

1 Introduction

The TPMC917-SW-82 Linux Device Driver consists of a full-duplex serial driver and a custom SRAM device driver, which allow the operation of a TPMC917 PMC device on Linux (Kernel 2.4.x+ and 2.6.x+) operating systems. These two drivers are connected through a HAL (Hardware Abstraction Layer) driver.

The TPMC917 Serial Device Driver is based on the standard Linux serial device driver and supports all standard terminal functions (TERMIOS).

Supported features of Serial Device Driver (TPMC917-10 only):

- Baudrates up to 115.2kbaud
- Each channel has a 64 Byte transmit and receive hardware FIFO
- Programmable trigger level for transmit and receive FIFO.
- Hardware (RTS/CTS) and software flow control (XON/XOFF) direct controlled by the serial controller. The advantage of this feature is that the transmission of characters will immediately stop as soon as a complete character is transmitted and not when the transmit FIFO is empty for handshake under software control. This will greatly improve flow control reliability.
- Designed as Linux kernel module with dynamically loading.
- Supports shared IRQ's.
- Build on new style PCI driver layout
- Creates a TTY device ttyTPMC917 and dialout device cuaTPMC917 (Kernel 2.4.x only) with dynamically allocated or fixed major device numbers.
- DEVFS support for automatic device node creation

Supported features of the SRAM Device Driver:

- Read variable sized data buffer from SRAM
- Write variable sized data buffer to SRAM
- map SRAM directly into userspace
- monitor battery backup
- wait for backup battery level fall to low

2 Installation

The directory TPMC917-SW-82 on the distribution media contains the following files:

TPMC917-SW-82.pdf	This manual in PDF format
TPMC917-SW-82-SRC.tar.gz	GZIP compressed archive with driver source code
Release.txt	Release information

The GZIP compressed archive TPMC917-SW-82-SRC.tar.gz contains the following files and directories:

Example/Makefile	Example application makefile
Example/tp917serexample.c	Send and receive example application
Example/tp917sersetspeed.c	Speed configuration example application
Example/tp917ramexample.c	RAM access example application
Example/tpmc917bist.c	Example for using Built-In-Self-Test
HAL/	Hardware abstraction layer driver needed for all kernel versions
HAL/Makefile	HAL driver makefile
HAL/tpmc917hal.c	HAL driver source file
HAL/tpmc917haldef.h	HAL driver private header file
HAL/tpmc917def.h	HAL driver public header file for Serial and Ram
HAL/tpxxxhwdep.c	HAL low level WINNT style hardware access functions source file
HAL/tpxxxhwdep.h	Access functions header file
Ram/	RAM driver directory
Ram/Makefile	RAM driver makefile
Ram/tpmc917ram.c	RAM driver source file
Ram/tpmc917ram.h	RAM driver application header file
Ram/tpmc917ramdef.h	RAM driver private header file
Ram/makenode	Shell script to create RAM device nodes without DEVFS
Serial/	UART driver directory
Serial/2.4.x	Kernel 2.4.x sources directory
Serial/2.4.x/Makefile	Serial driver makefile
Serial/2.4.x/tpmc917serial.c	Serial driver source file
Serial/2.4.x/tpmc917serialdef.h	Serial driver private header file
Serial/2.6.x	Kernel 2.6.x sources directory
Serial/2.6.x/Makefile	Serial driver makefile
Serial/2.6.x/tpmc917serial.c	Serial driver source file
Serial/2.6.x/tpmc917serialdef.h	Serial driver private header file
Serial/makenode	Shell script to create Serial device nodes without DEVFS
tpmc917ram.h	RAM driver application header file
tpmc917.h	Serial Driver header file

In order to perform an installation, extract all files of the archive TPMC917-SW-82-SRC.tar.gz to the desired target directory.

- Login as *root* and change to the target directory
- Copy tpmc917.h and tpmc917ram.h to */lib/modules/<version>/build/include* and */usr/include*

2.1 Build and install the device driver

- Login as *root*

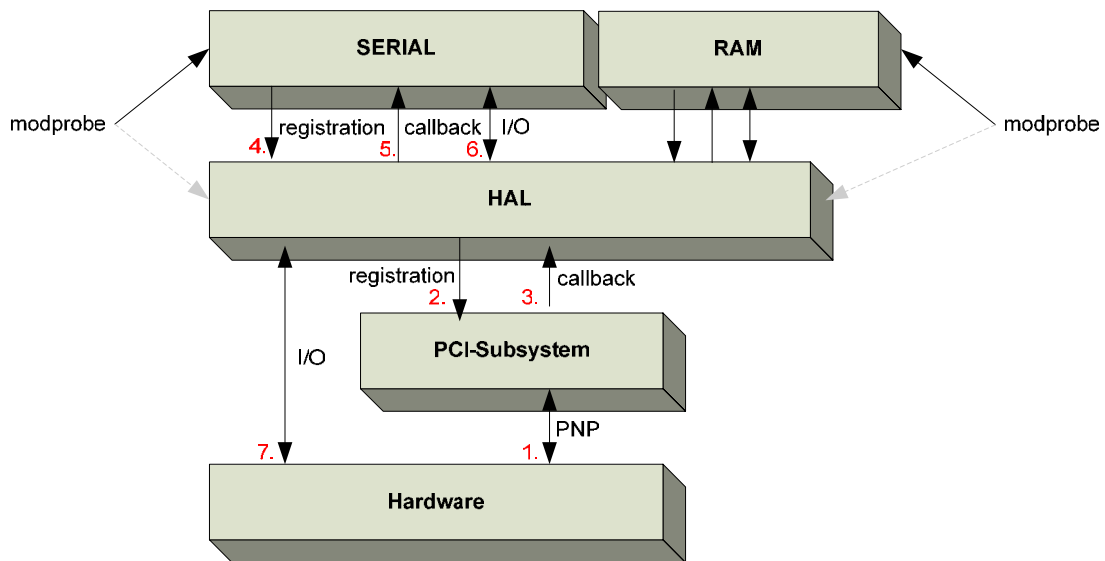
- Change to the HAL/ target directory
- To create and install the HAL driver in the module directory `/lib/modules/<version>/misc` enter:
make install
- Change to the Serial/<version> target directory
- To create and install the SERIAL driver in the module directory `/lib/modules/<version>/misc` enter:
make install
- Change to the Ram/ target directory
- To create and install the RAM driver in the module directory `/lib/modules/<version>/misc` enter:
make install
- To update the kernel's module dependencies, enter the following command:
depmod -aq

Some Linux distributions are using different declarations of the function `remap_page_range` with a differing number of parameters. If you get errors regarding this issue during compilation of `tpmc917ram.c`, change the corresponding macro definition (`TP_USE_REMAP_4PARAMS`) in this file.

2.2 Uninstall the device driver

- Login as `root`
- Change to the target directory
- To remove the driver from the module directory `/lib/modules/<version>/misc` , change to the directories `Serial/`, `Ram/` and `HAL/` and enter in each directory:
make uninstall

2.3 Install device driver into the running kernel



- To load the device drivers into the running kernel, login as root and execute one of the following commands. Both drivers will try to start the other remaining driver.
modprobe tpmc917ramdrv
modprobe tpmc917serialdrv
- After the first build or if you are using dynamic major device allocation it's necessary to create new device nodes on the file system. Please execute the script files *makenode*, which reside in Serial/ and Ram/ directory, to do this. If your kernel has enabled the device file system (devfs) then skip running the *makenode* script. Instead of creating device nodes from the script the driver itself takes creating and destroying of device nodes in its responsibility. Change to the directories Serial/ and Ram/ and enter in each directory:
sh makenode

On success the device driver will create a minor device for each functionality found (serial channels and SRAM). The first serial channel on the first PMC module (TPMC917-10 only) can be accessed by device node `/dev/ttySTPMC917_0`, the second channel by device node `/dev/ttySTPMC917_1` and so on. The first SRAM device node created is called `/dev/tpmc917ram_0`, a second device node will be named `/dev/tpmc917ram_1` and so on. The assignment of device nodes to physical PMC modules depends on the search order of the PCI bus driver.

2.4 Remove device driver from the running kernel

- To remove the device driver from the running kernel login as root and execute the following commands

```
# modprobe -r tpmc917ramdrv
# modprobe -r tpmc917serialdrv
```

If your kernel has enabled devfs, all `/dev/ttySTPMC917_*` and `/dev/tpmc917ram_*` nodes will be automatically removed from your file system after this.

Be sure that the driver isn't opened by any application program. If opened you will get the response "Device or resource busy" and the driver will still remain in the system until you close all opened files and execute `modprobe -r` again.

2.5 Change Major Device Numbers

This paragraph is only for Linux kernels without DEVFS installed.

The released TPMC917 driver (Serial + Ram) uses dynamic allocation of major device numbers. If this isn't suitable for the application it's possible to define a major number separately for the *TTY* and *CUA* driver as well as for the Ram driver.

To change the major number for the Serial device driver, edit the file `Serial/<version>/tpmc917serial.c`, change the following symbols to appropriate values and enter *make install* to create a new driver.

TPMC917_TTY_MAJOR Defines the value for the terminal device. Valid numbers are in range between 0 and 255. A value of 0 means dynamic number allocation.

TPMC917_CUA_MAJOR Defines the value for the dialout device. Valid numbers are in range between 0 and 255. A value of 0 means dynamic number allocation.

Example:

```
#define TPMC917_TTY_MAJOR      122
#define TPMC917_CUA_MAJOR     123
```

To change the major number of the Ram device driver, edit the file `Ram/tpmc917ramdef.h`, change the following symbol to an appropriate value and enter *make install* to create a new driver.

TPMC917_RAM_MAJOR Defines the value for the Ram device. Valid numbers are in range between 0 and 255. A value of 0 means dynamic number allocation.

Be sure that the desired major number isn't used by other drivers. Please check `/proc/devices` to see which numbers are free.

Keep in mind that it is necessary to create new device nodes if the major number for the TPMC917 driver has changed and the `makenode` script isn't used.

3 Serial Device Driver Programming

The TPMC917 Serial Device Driver loosely bases on the standard Linux terminal driver. Due to this way of implementation the driver interface and functionality is compatible to the standard Linux terminal driver.

Please refer to the TERMIOS man page and driver programming related man pages for more information about serial driver programming.

3.1 Simple Programming example

This example program opens the first serial channel of a TPMC917-10 device for read/write. After the device is open, it writes a "Hello World" string to the device and receives up to 80 bytes from the serial channel.

```
main()
{
    int fd;
    int count;
    char buffer[81];

    /* open the desired PMC device channel*/
    fd = open( "/dev/ttySTPMC917_0", O_RDWR | O_NOCTTY);

    if (fd < 0) exit(-1);

    /* write data to the certain channel */
    count = write(fd, "Hello World\n", 12);
    printf("%d bytes written\n", count);

    /* read up to 80 bytes from the device */
    count = read(fd, buffer, 80)
    if (count < 0) {
        printf("read error\n");
    }
    else {
        buffer[count] = 0;
        printf("%d bytes read <%s>\n", count, buffer);
    }

    close(fd);
}
```

The source files *tpmc917example.c* and *tpmc917setspeed.c* contains additional programming examples.

3.2 ioctl()

NAME

ioctl() device control functions

SYNOPSIS

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

```
int ioctl(int fildes, int request [, void *argp])
```

DESCRIPTION

The **ioctl** function sends a control code directly to a device, specified by *fildes*, causing the corresponding device to perform the requested operation. The argument *request* specifies the control code for the operation. The optional argument *argp* depends on the selected request and is described for each request in detail later in this chapter.

The following ioctl codes are defined in *tpmc917.h*:

Value	Meaning
<i>TPMC917_IOCQ_BIST</i>	Start Built-In-Self-Test

See below for more detailed information on each control code.

To use these TPMC917 specific control codes, the header file *tpmc917.h* must be included in the application.

RETURNS

On success, zero is returned. In case of an error, a value of -1 is returned. The global variable *errno* contains the detailed error code.

ERRORS

EINVAL	Invalid argument. This error code is returned if the requested ioctl function is unknown. Please check the argument <i>request</i> .
--------	--

Other function dependant error codes will be described for each ioctl code separately. Note, the TPMC917 driver always returns standard Linux error codes.

SEE ALSO

ioctl man pages

3.2.1 TPMC917_IOCQ_BIST

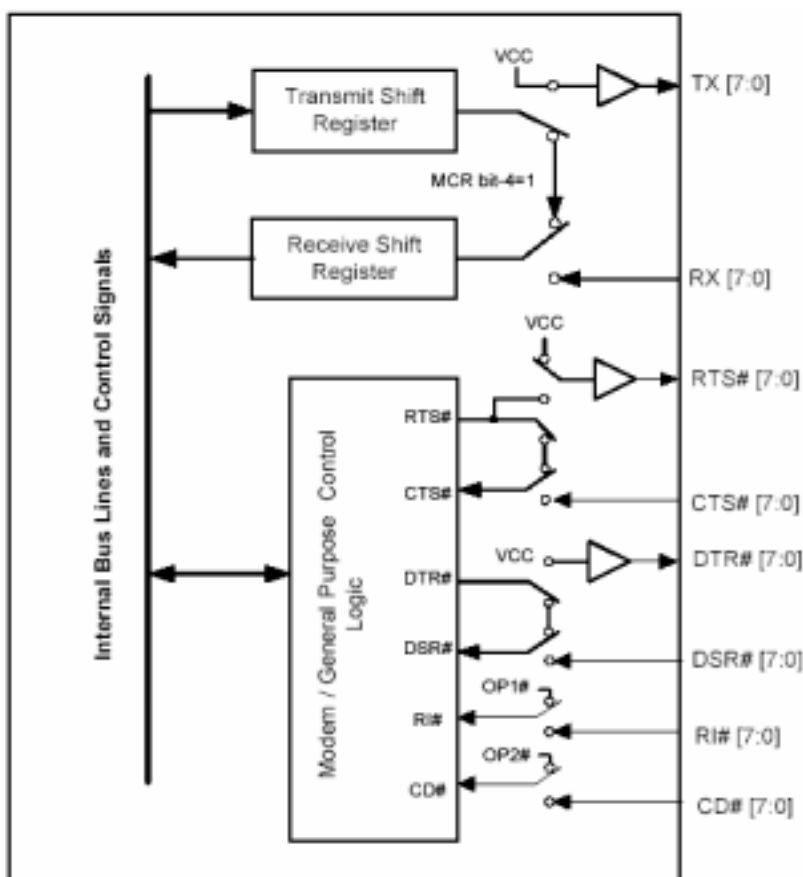
NAME

TPMC917_IOCQ_BIST – Start Built-In-Self-Test

DESCRIPTION

The TPMC917 Serial driver supports a special IOCTL function for testing the module hardware and for system diagnostics. The optional argument can be omitted for this ioctl function.

The functionality is called Built-In-Self-Test, or BIST. With BIST you can test each channel of all your modules separately. There are three different test classes. First is a line test, second an interrupt test and the last a data integrity test. All tests run with local channel loopback enabled, so you don't need an external cable connection.



The line test contains a test of all modem lines, although some of them are not available on the connectors, as you can see RTS and CTS, DTR and DSR, OP1 and RI and finally OP2 and CD. Only the static states for both electrical levels are tested on each sender – receiver line pair.

For testing interrupts the BIST transmits a test buffer with known data and size. All data should be received on the same channel during internal loopback. If not, there is an interrupt error. The buffer size is 1024 BYTE. The baudrate has to be set through the standard terminal IOCTL functions.

The last test verifies received data to assert data integrity.

EXAMPLE

```
/* Start Built-In Selftest, */
result = ioctl(tty1, TPMC917_IOCQ_BIST, NULL);

if (result) printf("Error during Built-In Selftest <%d, 0x%08X>!\n",
    result, result);
if (result < 0)
{
    printf("ERRNO %d - %s\n", errno, strerror(errno));
}
else if (result > 0)
{
    if (result & TPMC917_ERTSCTS)
        printf("RTS/CTS line broken!\n");
    if (result & TPMC917_EDTRDSR)
        printf("DTR/DSR line broken!\n");
    if (result & TPMC917_ERI)
        printf("OP1/RI line broken!\n");
    if (result & TPMC917_ECD)
        printf("OP2/DCD line broken!\n");
    if (result & TPMC917_EDATA)
        printf("Data integrity test failed!\n");
}
else
    printf("INFO: Port %s successfully tested.\n", DevName);
```

RETURNS

If return value is >0 one of three tests failed. Use the following flags to get a detailed error description.

TPMC917_ERTSCTS	If set RTS/CTS line broken.
TPMC917_EDTRDSR	If set DTR/DSR line broken.
TPMC917_ERI	If set OP1/RI line broken.
TPMC917_ECD	If set OP2/CD line broken.
TPMC917_EDATA	Data integrity test failed. No correct transmission possible.

ERRORS

ETIME	A timeout occurred during wait, interrupts do not work correctly.
EAGAIN	Your task should never been blocked. Change it to use the Built-In-Self-Test.
ERESTARTSYS	Interrupted by external signal.

4 SRAM Device Driver Programming

The TPMC917 SRAM Device Driver provides read/write access to the onboard SRAM, maps the SRAM directly into userspace and supplies battery monitoring facilities.

4.1 Device Input/Output functions

This chapter describes the interface to the device driver I/O system.

4.1.1 `open()`

NAME

`open()` opens a file descriptor.

SYNOPSIS

```
#include <fcntl.h>
```

```
int open ( const char *filename, int flags )
```

DESCRIPTION

The **open** function creates and returns a new file descriptor for the file named by *filename*. The *flags* argument controls how the file has to be opened. This is a bit mask. Create the value by the bitwise OR of the appropriate parameters (using the | operator in C). See also the GNU C Library documentation for more information about the open function and open flags.

EXAMPLE

```
int hCurrent;  
  
.  
.  
.  
  
hCurrent = open("/dev/tpmc917ram_0, O_RDWR);  
  
.  
.  
.
```

RETURNS

The usual return value from **open** is a non-negative integer file descriptor. In case of an error, a value of -1 is returned. The global variable *errno* contains the detailed error code.

ERRORS

E_NODEV The requested minor device does not exist.

This is the only error code returned by the driver, other codes may be returned by the I/O system during open. For more information about open error codes, see the *GNU C Library description – Low-Level Input/Output*.

SEE ALSO

GNU C Library description – Low-Level Input/Output

4.1.2 close()

NAME

close() closes a file descriptor.

SYNOPSIS

```
#include <unistd.h>
```

```
int close ( int filedes )
```

DESCRIPTION

The **close** function closes the file descriptor *filedes*.

EXAMPLE

```
int hCurrent;  
  
. . .  
  
if (close(hCurrent) != 0)  
{  
    * handle close error conditions */  
}
```

RETURNS

The usual return value from **close** is 0. In case of an error, a value of -1 is returned. The global variable *errno* contains the detailed error code.

ERRORS

E_NODEV The requested minor device does not exist.

This is the only error code returned by the driver, other codes may be returned by the I/O system during **close**. For more information about close error codes, see the *GNU C Library description – Low-Level Input/Output*.

SEE ALSO

GNU C Library description – Low-Level Input/Output

4.1.3 mmap()

NAME

mmap() maps kernel memory into userspace

SYNOPSIS

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

```
void *mmap ( void *addr, size_t len, int prot, int flags, int filedes, off_t off )
```

DESCRIPTION

The mmap() function establishes a mapping between a process' address space and a file or shared memory object.

PARAMETERS

addr

Specifies a preferred address where the memory should be mapped. If it is NULL, the address is determined automatically.

len

Size of memory to be mapped (in bytes). Should be 0x400000 for TPMC917 (4 MB) to map all available memory.

prot

Describes which operations can be performed on the mapped memory. Should be PROT_READ | PROT_WRITE to allow both read and write operations.

flags

Describes the mapping flags. Should be MAP_FILE | MAP_SHARED.

filedes

File descriptor to RAM device previously opened by a call to open().

off

Offset relative to the beginning of the memory section. Should be 0 to map all available memory.

EXAMPLE

```
int hCurrent;
unsigned char* pSram;

pSram = mmap( NULL,                /* preferred address */
              0x400000,            /* size of memory to be mapped */
              PROT_READ | PROT_WRITE, /* memory protection */
              MAP_FILE | MAP_SHARED, /* flags */
              hCurrent,           /* file descriptor */
              0);                /* offset */
if (pSram == MAP_FAILED)
{
    /* handle error */
}
```

RETURNS

The usual return value of **mmap()** is a pointer which can be used by the application to directly access the memory. In case of an error, **MAP_FAILED** is returned. The global variable *errno* contains the detailed error code.

ERRORS

Error codes may be returned by the I/O system during **mmap**. For more information about mmap error codes, please refer to the manpage of mmap and the detailed description of Linux' memory management functions (mman).

SEE ALSO

GNU C Library description – Low-Level Input/Output

4.1.4 munmap()

NAME

`munmap()` unmap pages of memory

SYNOPSIS

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

```
int munmap ( void *addr, size_t len )
```

DESCRIPTION

The `munmap()` function removes the mappings for pages in the specified range.

PARAMETERS

addr

Specifies the address returned by a previous call to `mmap()`.

len

Size of memory mapped by a previous call to `mmap()`.

EXAMPLE

```
int hCurrent;
int result;
unsigned char* pSram;

result = munmap( pSram,          /* address returned by mmap() */
                0x400000);      /* size of mapped memory      */
if (result < 0)
{
    /* handle error */
}
```

RETURNS

Upon successful completion, `munmap()` returns 0; otherwise, it returns -1. The global variable `errno` contains the detailed error code.

ERRORS

`EINVAL` The specified arguments are invalid.
Other error codes may be returned by the I/O system during `munmap`.

SEE ALSO

GNU C Library description – Low-Level Input/Output

4.1.5 ioctl()

NAME

ioctl() device control functions

SYNOPSIS

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

```
int ioctl ( int filedes, int request [, void *argp] )
```

DESCRIPTION

The **ioctl** function sends a control code directly to a device, specified by *filedes*, causing the corresponding device to perform the requested operation.

The argument *request* specifies the control code for the operation. The optional argument *argp* depends on the selected request and is described for each request in detail later in this chapter.

The following ioctl codes are defined in *tpmc917ram.h*:

Value	Meaning
<i>TP917RAM_IOCTL_GETMEMSIZE</i>	Get size of TPMC917 onboard memory.
<i>TP917RAM_IOCTL_READ</i>	Read from TPMC917 onboard memory.
<i>TP917RAM_IOCTL_WRITE</i>	Write to TPMC917 onboard memory.
<i>TP917RAM_IOCTL_GETBATSTAT</i>	Get state of TPMC917 onboard backup battery.
<i>TP917RAM_IOCTL_WAITBATLOW</i>	Wait until the state of the backup battery falls to low.

See below for more detailed information on each control code.

To use these TPMC917 specific control codes the header file *tpmc917ram.h* must be included in the application.

RETURNS

On success, zero is returned. In case of an error, a value of -1 is returned. The global variable *errno* contains the detailed error code.

ERRORS

EINVAL Invalid argument. This error code is returned if the requested ioctl function is unknown. Please check the argument *request*.

Other function dependant error codes will be described for each ioctl code separately. Note, the TPMC917 RAM driver always returns standard Linux error codes.

SEE ALSO

ioctl man pages

4.1.5.1 TP917RAM_IOCTL_GETMEMSIZE

NAME

TP917RAM_IOCTL_GETMEMSIZE Get size of TPMC917 onboard memory.

DESCRIPTION

This TPMC917 SRAM ioctl function returns the SRAM's memory size located on the TPMC917 module. A pointer to an *unsigned long* value must be supplied to the device driver, where the size is returned.

EXAMPLE

```
#include "tpmc917ram.h"

int          hCurrent;
int          result;
unsigned long MemorySize;

. . .

result = ioctl( hCurrent, TP917RAM_IOCTL_GETMEMSIZE, &MemorySize );
if (result >= 0)
{
    printf("\nMemory Size = %ld (0x%X)\n", MemorySize, MemorySize);
}
else
{
    printf("\nGet Memory Size failed --> Error = %d\n",
        errno);
}
```

ERRORS

EFAULT Invalid pointer to the parameter buffer. Please check the argument *argp*.
All other returned error codes are system error conditions.

SEE ALSO

ioctl man pages

4.1.5.2 TP917RAM_IOCTL_READ

NAME

TP917RAM_IOCTL_READ Read from TPM917 onboard memory.

DESCRIPTION

This TPM917 SRAM ioctl function writes a dynamically adjustable data buffer into the onboard SRAM. No data verification is performed. The parameter *argp* passes a pointer to a TP917RAM_MEMIO_BUF structure to the device driver.

The TP917RAM_MEMIO_BUF structure has the following layout:

```
typedef struct
{
    unsigned long    offset;
    unsigned long    size;
    unsigned char    pData[1];          /* dynamically expandable */
} TP917RAM_MEMIO_BUF;
```

Members

offset

Specifies the offset relative to the beginning of the SRAM, from where the data is retrieved. The SRAM is treated like a linear contiguous memory block

size

Specifies the number of bytes to read. The data buffer must be large enough to hold the amount of data.

pData

Specifies the data buffer, which can be dynamically enlarged.

EXAMPLE

```
#include "tpmc917ram.h"

int          hCurrent;
int          result;
unsigned long    totalSize;
TP917RAM_MEMIO_BUF    *pMemIoBuf;

/*
** allocate enough memory and read 20 bytes from SRAM, Offset=0
*/
```

```
totalSize = sizeof(TP917RAM_MEMIO_BUF) + 20*sizeof(unsigned char);
pMemIoBuf = (TP917RAM_MEMIO_BUF*)malloc( totalSize );
memset( pMemIoBuf, 0, totalSize );

pMemIoBuf->size = 20;
pMemIoBuf->offset = 0;
result = ioctl( hCurrent, TP917RAM_IOCX_READ, pMemIoBuf );
if (result >= 0)
{
    /* data successfully read */
    printf( "Data='%s'\n", pMemIoBuf->pData );
}
else
{
    printf("\nRead from memory failed --> Error = %d\n",
        errno);
}
```

ERRORS

EFAULT Invalid pointer to the parameter buffer. Please check the argument *argp*.
All other returned error codes are system error conditions.

SEE ALSO

ioctl man pages

4.1.5.3 TP917RAM_IOCTL_WRITE

NAME

TP917RAM_IOCTL_WRITE Write to TPM917 onboard memory.

DESCRIPTION

This TPM917 SRAM ioctl function writes a dynamically adjustable data buffer into the onboard SRAM. No data verification is performed. The parameter *argp* passes a pointer to a TP917RAM_MEMIO_BUF structure to the device driver.

The TP917RAM_MEMIO_BUF structure has the following layout:

```
typedef struct
{
    unsigned long    offset;
    unsigned long    size;
    unsigned char    pData[1];          /* dynamically expandable */
} TP917RAM_MEMIO_BUF;
```

Members

offset

Specifies the offset relative to the beginning of the SRAM, where the data is to be written. The SRAM is treated like a linear contiguous memory block

size

Specifies the number of bytes to be written. The data buffer must be large enough to hold the amount of data.

pData

Specifies the data buffer, which can be dynamically enlarged.

EXAMPLE

```
#include "tpmc917ram.h"

int          hCurrent;
int          result;
unsigned long totalSize;
TP917RAM_MEMIO_BUF *pMemIoBuf;

/*
** allocate enough memory and read 20 bytes from SRAM, Offset=0
*/
```

```
totalSize = sizeof(TP917RAM_MEMIO_BUF) + 20*sizeof(unsigned char);
pMemIoBuf = (TP917RAM_MEMIO_BUF*)malloc( totalSize );
memset( pMemIoBuf, 0, totalSize );

pMemIoBuf->size = 20;
pMemIoBuf->offset = 0;
result = ioctl( hCurrent, TP917RAM_IOCX_READ, pMemIoBuf );
if (result >= 0)
{
    /* data successfully read */
    printf( "Data='%s'\n", pMemIoBuf->pData );
}
else
{
    printf("\nRead from memory failed --> Error = %d\n",
        errno);
}
```

ERRORS

EFAULT Invalid pointer to the parameter buffer. Please check the argument *argp*.
All other returned error codes are system error conditions.

SEE ALSO

ioctl man pages

4.1.5.4 TP917RAM_IOCQ_GETBATSTAT

NAME

TP917RAM_IOCQ_GETBATSTAT Get size of TPMC917 onboard memory.

DESCRIPTION

This TPMC917 SRAM ioctl function returns the state of the SRAM's battery backup. The parameter *argp* passes a pointer to an *unsigned long* value to the device driver. Possible returned values are *TP917_BATTERY_LOW* and *TP917_BATTERY_OK* (see *tpmc917ram.h*).

EXAMPLE

```
#include "tpmc917ram.h"

int          hCurrent;
int          result;
unsigned long BatteryState;

result = ioctl( hCurrent, TP917RAM_IOCQ_GETBATSTAT, &BatteryState );
if (result >= 0)
{
    if (BatteryState == TP917_BATTERY_LOW)
    {
        printf( "BATTERY LOW!!! \n" );
    } else {
        printf( "Battery OK.\n" );
    }
}
else
{
    printf("\nGet Battery State failed --> Error = %d\n",
        errno);
}
```

ERRORS

EFAULT Invalid pointer to the parameter buffer. Please check the argument *argp*.
All other returned error codes are system error conditions.

SEE ALSO

ioctl man pages

4.1.5.5 TP917RAM_IOC_WAITBATLOW

NAME

TP917RAM_IOC_WAITBATLOW Wait until the state of the backup battery falls to low.

DESCRIPTION

This TPMC917 SRAM ioctl function waits indefinitely until the SRAM's battery backup state turns to low. No timeout can be specified, so this function should be called out of a dedicated battery monitoring thread. No additional parameter is necessary.

EXAMPLE

```
#include "tpmc917ram.h"

int                hCurrent;
int                result;

result = ioctl( hCurrent, TP917RAM_IOC_WAITBATLOW, 0 );
if (result >= 0)
{
    printf( "BATTERY LOW!!! \n" );
}
else
{
    printf("\nWait for Battery Low failed --> Error = %d\n",
        errno);
}
```

ERRORS

All returned error codes are system error conditions.

SEE ALSO

ioctl man pages

5 Diagnostic

If the TPMC917 driver does not work properly it is helpful to get some status information from the driver respective kernel.

The Linux `/proc` file system provides information about kernel, resources, drivers, devices and so on. The following screen dumps display information of a correct running TPMC917 driver (see also the `proc` man pages).

```
# cat /proc/modules
tpmc917serialdrv      21964    0 (autoclean) (unused)
tpmc917ramdrv         10128    0 (unused)
tpmc917haldrv         24108    0 [tpmc917serialdrv tpmc917ramdrv]
mousedev              5428     1 (autoclean)
input                  5792     0 (autoclean) [mousedev]
parport_pc            18756     1 (autoclean)
lp                     8868     0 (autoclean)
parport                36480     1 (autoclean) [parport_pc lp]
autofs                 12948     0 (autoclean) (unused)

# cat /proc/tty/driver/tpmc917serial
TEWS TECHNOLOGIES - TPMC917 UART driver (Kernel 2.4.x): 1.0.0 revision:
2005-05-26
0: uart:ST16C654 port:E0897000 irq:5 tx:0 rx:0
1: uart:ST16C654 port:E0897008 irq:5 tx:0 rx:0
2: uart:ST16C654 port:E0897010 irq:5 tx:0 rx:0
3: uart:ST16C654 port:E0897018 irq:5 tx:0 rx:0
4: uart:ST16C654 port:E0899000 irq:11 tx:0 rx:0
5: uart:ST16C654 port:E0899008 irq:11 tx:0 rx:0
6: uart:ST16C654 port:E0899010 irq:11 tx:0 rx:0
7: uart:ST16C654 port:E0899018 irq:11 tx:0 rx:0

# cat /proc/tty/drivers
tpmc917serial      /dev/cuaTPMC917 252 0-127 serial:callout
tpmc917serial      /dev/ttyTPMC917 253 0-127 serial
serial              /dev/cua          5 64-127 serial:callout
serial              /dev/ttyS         4 64-127 serial
pty_slave           /dev/pts         143 0-255 pty:slave
pty_master          /dev/ptm         135 0-255 pty:master
pty_slave           /dev/pts         142 0-255 pty:slave
pty_master          /dev/ptm         134 0-255 pty:master
pty_slave           /dev/pts         141 0-255 pty:slave
pty_master          /dev/ptm         133 0-255 pty:master
pty_slave           /dev/pts         140 0-255 pty:slave
pty_master          /dev/ptm         132 0-255 pty:master
pty_slave           /dev/pts         139 0-255 pty:slave
pty_master          /dev/ptm         131 0-255 pty:master
pty_slave           /dev/pts         138 0-255 pty:slave
pty_master          /dev/ptm         130 0-255 pty:master
pty_slave           /dev/pts         137 0-255 pty:slave
pty_master          /dev/ptm         129 0-255 pty:master
pty_slave           /dev/pts         136 0-255 pty:slave
pty_master          /dev/ptm         128 0-255 pty:master
pty_slave           /dev/ttyp         3 0-255 pty:slave
pty_master          /dev/pty          2 0-255 pty:master
/dev/vc/0           /dev/vc/0        4      0 system:vtmaster
/dev/ptmx           /dev/ptmx         5      2 system
/dev/console        /dev/console      5      1 system:console
/dev/tty            /dev/tty          5      0 system:/dev/tty
unknown             /dev/vc/%d        4 1-63 console
```

```
# cat /proc/interrupts
          CPU0
0:      151112      XT-PIC  timer
1:         6      XT-PIC  keyboard
2:         0      XT-PIC  cascade
5:         0      XT-PIC  TPMC917RAM, TPMC917, TPMC917, TPMC917,
TPMC917
8:         1      XT-PIC  rtc
11:     3475      XT-PIC  eth0, TPMC917RAM, TPMC917, TPMC917,
TPMC917, TPMC917
12:         47      XT-PIC  PS/2 Mouse
14:     13085      XT-PIC  ide0
15:         0      XT-PIC  ide1
NMI:         0
ERR:         0
```

```
# lspci -v
```

```
...
```

```
00:0a.0 Class 0808: TEWS Datentechnik GmbH: Unknown device 0395 (rev 0a)
Subsystem: TEWS Datentechnik GmbH: Unknown device 000a
Flags: medium devsel, IRQ 5
Memory at eb823000 (32-bit, non-prefetchable) [size=128]
Memory at eb000000 (32-bit, non-prefetchable) [size=4M]
Memory at eb821000 (32-bit, non-prefetchable) [size=64]
```

```
00:0b.0 Class 0808: TEWS Datentechnik GmbH: Unknown device 0395 (rev 0a)
Subsystem: TEWS Datentechnik GmbH: Unknown device 000a
Flags: medium devsel, IRQ 11
Memory at eb822000 (32-bit, non-prefetchable) [size=128]
Memory at eb400000 (32-bit, non-prefetchable) [size=4M]
Memory at eb824000 (32-bit, non-prefetchable) [size=64]
```

```
...
```