The Embedded I/O Company

TEWS TECHNOLOGIES

TPMC917-SW-42

VxWorks Device Driver

4 MB SRAM with Battery Backup and

4 Channel Serial Interface PMC

Version 2.1.x

User Manual

Issue 2.1.0

June 2009

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

TEWS TECHNOLOGIES LLC
9190 Double Diamond Parkway,
Suite 127, Reno, NV 89521, USA
www.tews.com
Phone: +1 (775) 850 5830
Fax: +1 (775) 201 0347
e-mail: usasales@tews.com

TPMC917-SW-42

VxWorks Device Driver

4MB (2MB) SRAM with Battery Backup and
4 Channel Serial Interface PMC

Supported Modules:
TPMC917

Issue	Description	Date
1.0	First Issue	September 2000
1.1	General Revision	January 2004
2.0.0	SRAM access functions added, callback functions removed	May 2005
2.1.0	General Revision, new address TEWS LLC, filelist modified	June 3, 2009

Table of Contents

1 Introduction

1.1 Device Driver

The TPMC917-SW-42 VxWorks device driver software allows the operation of the supported TPMC917 conforming to the VxWorks I/O system specification. This includes a device-independent basic I/O interface with *open*(), *close(), read(), write(),* and *ioctl()* functions and a buffered I/O interface (*fopen()*, *printf()*, *scanf()*, ...).

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.

The TPMC917-SW-42 device driver supports the following features:

- ring buffering of input and output
- raw mode
- optional line mode with backspace and line-delete functions
- optional processing of X-on/X-off
- optional RETURN/LINEFEED conversion
- optional echoing of input characters
- optional stripping of the parity bit from 8 bit input
- optional special characters for shell abort and system restart
- select FIFO triggering point
- use 5..8 bit data words
- use 1, 1.5 or 2 stop bits
- optional even or odd parity
- enable/disable hardware handshake (only in FIFO mode)

The following functions are supported for battery control:

- wait for battery low event
- poll the battery state

The following functions are supported for SRAM access:

- Read a variable data buffer from the SRAM
- Write a variable data buffer to the SRAM

The TPMC917-SW-42 supports the modules listed below:

TPMC917-10	4 MB SRAM module with battery backup and 4 channel RS232	(PMC)
TPMC917-20	4 MB SRAM module with battery backup	(PMC)
TPMC917-21	2 MB SRAM module with battery backup	(PMC)

In this document all supported modules and devices will be called TPMC917. Specials for certain devices will be advised.

To get more information about the features and use of supported devices it is recommended to read the manuals listed below.

TPMC917 User Manual

TPMC917 Engineering Manual

2 Installation

Following files are located on the distribution media:

Directory path 'TPMC917-SW-42':

tpmc917drv.c	TPMC917 device driver source
tpmc917def.h	TPMC917 driver include file
tpmc917.h	TPMC917 include file for driver and application
tpmc917pci.c	TPMC917 PCI MMU mapping for Intel x86 based targets
tpmc917exa.c	Example application
include/tdhal.h	Hardware dependent interface functions and definitions
TPMC917-SW-42-2.1.0.pdf	PDF copy of this manual
ChangeLog.txt	Release history
Release.txt	Release information

2.1 Include device driver in VxWorks projects

For including the TPMC917-SW-42 device driver into a VxWorks project (e.g. Tornado IDE or Workbench) follow the steps below:

(1) Copy the files from the distribution media into a subdirectory in your project path. (For example: ./TPMC917)

(2) Add the device drivers C-files to your project.

(3) Now the driver is included in the project and will be built with the project.

For a more detailed description of the project facility please refer to your VxWorks User's Guide (e.g. Tornado, Workbench, etc.)

2.2 Special installation for Intel x86 based targets

The TPMC917 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_FAMILY**. If the content of this macro is equal to *I80X86* 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, the required device memory spaces can't be accessed.

To solve this problem a MMU mapping entry has to be added for the required TPMC917 PCI memory spaces prior the MMU initialization (*usrMmuInit()*) is done.

The C source file **tpmc917pci.c** contains the function *tpmc917PciInit().* This routine finds out all TPMC917 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 (*usrMmuInit()*).

The right place to call the function *tpmc917PciInit()* is at the end of the function *sysHwInit()* in **sysLib.c** (it can be opened from the project *Files* window).

Be sure that the function is called prior to MMU initialization otherwise the TPMC917 PCI spaces remains unmapped and an access fault occurs during driver initialization.

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

```
tpmc917PciInit();
```

Modifying the sysLib.c file will change the sysLib.c in the BSP path. Remember this for future projects and recompilations.

2.3 BSP dependent adjustments

The driver includes a file called *include/tdhal.h* which contains functions and definitions for BSP adaptation. It may be necessary to modify them for BSP specific settings. Most settings can be made automatically by conditional compilation set by the BSP header files, but some settings must be configured manually. There are two way of modification, first you can change the *include/tdhal.h* and define the corresponding definition and its value, or you can do it, using the command line option *–D*.

There are 3 offset definitions (*USERDEFINED_MEM_OFFSET*, *USERDEFINED_IO_OFFSET*, and *USERDEFINED_LEV2VEC*) that must be configured if a corresponding warning message appears during compilation. These definitions always need values. Definition values can be assigned by command line option *-D<definition>=<value>*.

Definition	Description
USERDEFINED_MEM_OFFSET	The value of this definition must be set to the offset between CPU-Bus and PCI-Bus Address for PCI memory space access
USERDEFINED_IO_OFFSET	The value of this definition must be set to the offset between CPU-Bus and PCI-Bus Address for PCI I/O space access
USERDEFINED_LEV2VEC	The value of this definition must be set to the difference of the interrupt vector (used to connect the ISR) and the interrupt level (stored to the PCI header)

Another definition allows a simple adaptation for BSPs that utilize a *pciIntConnect()* function to connect shared (PCI) interrupts. If this function is defined in the used BSP, the definition of *USERDEFINED_SEL_PCIINTCONNECT* should be enabled. The definition by command line option is made by *-D<definition>.*

Please refer to the BSP documentation and header files to get information about the interrupt connection function and the required offset values.

2.4 System resource requirement

The table gives an overview over the system resources that will be needed by the driver.

Resource	Driver requirement	Devices requirement
Memory	< 1 KB	< 1 KB
Stack	< 1 KB	---
Semaphores	1	1

Memory and Stack usage may differ from system to system, depending on the used compiler and its setup.

The following formula shows the way to calculate the common requirements of the driver and devices.

*<total requirement> = <driver requirement> + (<number of devices> * <device requirement>)*

The maximum usage of some resources is limited by adjustable parameters. If the application and driver exceed these limits, increase the according values in your project.

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 tp917Drv()

NAME

tp917Drv() - install the TPMC917 driver in the I/O system

SYNOPSIS

#include “tpmc917.h”

STATUS tp917Drv(void)

DESCRIPTION

This function searches for devices on the PCI bus, and installs the TPMC917 driver in the I/O system.

A call to this function is the first thing the user has to do before adding any device to the system or performing any I/O request.

EXAMPLE

```
#include "tpmc917.h”

STATUS            result;

/*-------------------
  Initialize Driver
  -------------------*/
result = tp917Drv();
if (result == ERROR)
{
     /* Error handling */
}
```

RETURNS

OK or ERROR. If the function fails an error code will be stored in *errno*.

ERROR CODES

Error codes are only set by system functions. The error codes are stored in *errno* and can be read with the function *errnoGet()*.

SEE ALSO

VxWorks Programmer's Guide: I/O System

3.2 tp917SerDevCreate()

NAME

tp917SerDevCreate() - add a serial TPMC917 device to the system and initializes device hardware with the selected configuration for this serial port

SYNOPSIS

```
STATUS tp917SerDevCreate
(
    char            *name,      /* name of the device to create          */
    int             channel,    /* physical channel for this device      */
    int             rdBufSize,  /* read buffer size in bytes             */
    int             wrtBufSize, /* write buffer size in bytes            */
    TP917_CHANCONF  *devconf    /* device configuration                  */
)
```

DESCRIPTION

This routine creates a device on a specified serial channel that will be serviced by the TPMC917 driver. It is only available for TPMC917-10. This function must be called before performing any I/O request to this device.

PARAMETERS

name

This parameter supplies a null-terminated ASCII string, which describes the name of the device which should be created (e.g. "tp917/0")

channel

This parameter specifies the global channel number of the serial port which should be connected to the specified device name. The channel number is counted from 0 for all found TPMC917 serial channels.

rdBufSize

This parameter specifies the size of the internal read buffer for this serial port.

wrtBufSize

This parameter specifies the size of the internal write buffer for this serial port.

devconf

This parameter points to a configuration structure. The configuration structure is compatible to the default structure in the *tpmc917.h* file. If the pointer is set to zero when calling this function, the default parameters specified in *tpmc917def.h* will be used.

The structure has the following layout:

```
typedef struct
{
        unsigned int    Baudrate;
        unsigned int    Options;
        unsigned int    FifoTrigRcv;
        unsigned int    FifoTrigTrm;
        unsigned int    Databits;
        unsigned int    Stopbits;
        unsigned int    Parity;
        unsigned int    Handshake;
} TP917_CHANCONF;
```

Baudrate

Select the initial baud rate of the channel - allowed values are (50 … 115200)

Options

Select the initial VxWorks driver options

FifoTrigRcv

Select the receiver FIFO trigger level. If one of the FIFO trigger levels is set to *TP917F_NO*, the other level is also disabled.

FifoTrigTrm

Select the transmitter FIFO trigger level. If one of the FIFO trigger levels is set to *TP917F_NO*, the other level is also disabled.

Databits

Select the number of data bits in one data word. Use the predefined values in *tpmc917.h.*

Stopbits

Selects the number of stop bits in one data word. Use the predefined values in *tpmc917.h.*

Parity

Select the parity checking mode - use the predefined values in *tpmc917.h*

Handshake

If hardware handshake is enabled or disabled (after startup). Select YES to use the handshake lines, otherwise NO.

EXAMPLE

```
#include "tpmc917.h"

TP917_CHANCONF     tp917setup =
{
     115200,                                  /* 115200 Baud              */
     OPT_TERMINAL,                            /* terminal options         */
     TP917F_NO, TP917F_NO,                    /* no FIFO                  */
     TP917DB_8, TP917SB_10, TP917NOP,         /* 8/1/0 Data/Stop/Parity   */
     NO                                       /* no handshake             */
};


/*-------------------------------------------------------------
  Create the device "/tp917/0" on channel 0 with read and write
  buffer sizes of 512 bytes.
    Baudrate:  115200Baud
    Options:   Terminal
    FIFOs:     disabled
    Databits:  8
    Stopbits:  1
    Parity:    off
    Handshake: off
  -------------------------------------------------------------*/
status = tp917SerDevCreate ("/tp917/0", 0, 512, 512, &tp917setup);
```

RETURNS

OK or ERROR (if the driver is not installed, or the channel is invalid or the device already exists)

3.3 tp917SramDevCreate()

NAME

tp917SramDevCreate() - add a device for TPMC917 SRAM.

SYNOPSIS

```
STATUS tp917SramDevCreate
(
    char        *name,                  /* name of the device to create     */
    int         devno                   /* physical device number           */
)
```

DESCRIPTION

This routine creates a device on a specified serial channel that will be serviced by the TPMC917 driver. This function must be called before performing any I/O request to this device.

PARAMETERS

name

This parameter supplies a null-terminated ASCII string, which describes the name of the device which should be created (e.g. "tp917/sram0")

devno

This parameter specifies the global device number of the found TPMC917 devices. The device number is counted from 0 for all found TPMC917 modules.

EXAMPLE

```
#include  "tpmc917.h”

/*-------------------------------------------------------
  Create the device "/tp917/sram0" for device 0
  -------------------------------------------------------*/
status = tp917SramDevCreate ("/tp917/sram0", 0);
```

RETURNS

OK or ERROR (if the driver is not installed, or the channel is invalid or the device already exists)

3.4 tp917PciInit()

NAME

tp917PciInit() – Generic PCI device initialization

SYNOPSIS

void tp917PciInit()

DESCRIPTION

This function is required only for Intel x86 VxWorks platforms. The purpose is to setup the MMU mapping for all required TPMC917 PCI spaces (base address register) and to enable the TPMC917 device for access.

The global variable *tp917Status* obtains the result of the device initialization and can be polled later by the application before the driver will be installed.

Value	Meaning
> 0	Initialization successful completed. The value of tp917Status is equal to the number of mapped PCI spaces
0	No TPMC917 device found
< 0	Initialization failed. The value of (tp917Status & 0xFF) is equal to the number of mapped spaces until the error occurs. Possible cause: Too few entries for dynamic mappings in sysPhysMemDesc[]. Remedy: Add dummy entries as necessary (syslib.c).

EXAMPLE

```
extern void tp917PciInit();

…

tp917PciInit();
```

4 I/O Functions

4.1 open()

NAME

open() - open a device or file.

SYNOPSIS

```
int open
(
        const char   *name,
        int          flags,
        int          mode
)
```

DESCRIPTION

Before I/O can be performed to the TPMC917 device, a file descriptor must be opened by invoking the basic I/O function *open().*

PARAMETER

name

Specifies the device which shall be opened, the name specified in tp917DevCreate() must be used

flags

Not used

mode

Not used

EXAMPLE

```
int      fd;

/*----------------------------------------
  Open the device named "/tp917/0" for I/O
  ----------------------------------------*/
fd = open("/tp917/0", 0, 0);
if (fd == ERROR)
{
     /* Handle error */
}
```

RETURNS

A device descriptor number or ERROR. If the function fails an error code will be stored in *errno*.

ERROR CODES

The error code can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual).

SEE ALSO

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

4.2 close()

NAME

close() – close a device or file

SYNOPSIS

```
STATUS close
(
     int          fd
)
```

DESCRIPTION

This function closes opened devices.

PARAMETER

fd

This file descriptor specifies the device to be closed. The file descriptor has been returned by the *open()* function.

EXAMPLE

```
int       fd;
STATUS    retval;

/*----------------
  close the device
  ----------------*/
retval = close(fd);
if (retval == ERROR)
{
     /* Handle error */
}
```

RETURNS

OK or ERROR. If the function fails, an error code will be stored in *errno*.

ERROR CODES

The error code can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual).

SEE ALSO

ioLib, basic I/O routine - close()

4.3 read()

NAME

read() – read data from a specified device.

SYNOPSIS

```
int read
(
        int             fd,
        char            *buffer,
        size_t          maxbytes
)
```

DESCRIPTION

This function can be used to read data from the device.

PARAMETER

fd

This file descriptor specifies the device to be used. The file descriptor has been returned by the *open()* function.

buffer

This argument points to a user supplied buffer. The returned data will be filled into this buffer.

maxbytes

This parameter specifies the maximum number of read bytes (buffer size).

EXAMPLE

```
#define   BUFSIZE   80

int           fd;
char          buffer[BUFSIZE];
int           retval;

/*------------------------------------------------------------
  Read up to 80 bytes from the serial channel connected with
  the device descriptor fd
  ------------------------------------------------------------*/
retval = read(fd, buffer, BUFSIZE);
if (retval != ERROR)
{
     printf("%d bytes read\n", retval);
}
else
{
     /* handle the read error */
}
```

RETURNS

Number of bytes read or ERROR. If the function fails an error code will be stored in *errno*.

ERROR CODES

The error code can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual).

SEE ALSO

ioLib, basic I/O routine - read()

4.4 write()

NAME

write() – write data from a buffer to a specified device.

SYNOPSIS

```
int write
(
        int             fd,
        char            *buffer,
        size_t          nbytes
)
```

DESCRIPTION

This function can be used to write data to the device.

PARAMETER

fd

This file descriptor specifies the device to be used. The file descriptor has been returned by the *open()* function.

buffer

This argument points to a user supplied buffer. The data of the buffer will be written to the device.

nbytes

This parameter specifies the number of bytes to be written.

EXAMPLE

```
int           fd;
char          buffer[] = “Hello World”;
int           retval;

/*------------------------------
  Write data to a TPMC917 device
  ------------------------------*/
retval = write(fd, buffer, strlen(buffer));
if (retval != ERROR)
{
     printf(“%d bytes written\n”, retval);
}
else
{
     /* handle the write error */
}
```

RETURNS

Number of bytes written or ERROR. If the function fails an error code will be stored in *errno*.

ERROR CODES

The error code can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual).

SEE ALSO

ioLib, basic I/O routine - write()

4.5 ioctl()

NAME

ioctl() - performs an I/O control function.

SYNOPSIS

```
#include <stdio.h>
#include <tyLib.h>
#include <ioLib.h>
#include “tpmc917.h”

int ioctl
(
    int     fd,
    int     request,
    int     arg
)
```

DESCRIPTION

Special I/O operation that do not fit to the standard basic I/O calls (read, write) will be performed by calling the ioctl() function.

PARAMETER

fd

This file descriptor specifies the device to be used. The file descriptor has been returned by the *open()* function.

request

This argument specifies the function that shall be executed. Following functions are defined:

Function	Description
FIOBAUDRATE	Configure Serial Baudrate
FIOFIFO	Configure FIFO
FIODATABITS	Configure number of Data Bits
FIOSTOPBITS	Configure number of Stop Bits
FIOPARITY	Configure Parity mode
FIOENABLEHWHS	Enable hardware handshake
FIODISABLEHWHS	Disable hardware handshake
FIOSETBREAK	Generate BREAK signal
FIOCLEARBREAK	Clear BREAK signal
FIOCHECKBREAK	Check for BREAK signal

FIOCHECKERRORS	Check for errors
FIORECONFIGURE	Reconfigure serial channel
FIOGETBATSTAT	Get Battery state
FIOWAITBATLOW	Wait for Battery Low Event
FIOSRAMREAD	Read data from SRAM
FIOSRAMWRITE	Write data to SRAM
FIOSRAMGETSIZE	Get SRAM size
FIOSRAMGETPTR	Get pointer to SRAM area

arg

This parameter depends on the selected function (request). How to use this parameter is described below with the function.

RETURNS

OK or ERROR. If the function fails an error code will be stored in *errno*.

ERROR CODES

The error code can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual). Function specific error codes will be described with the function.

SEE ALSO

ioLib, basic I/O routine - ioctl()

4.5.1 FIOBAUDRATE

The *FIOBAUDRATE* function is a standard function with a few points to pay attention to. The selected baud rate is always set to the next selectable value. This ioctl function is only available for serial channel devices.

The selectable baud rates are calculated with the following formula:

Baudrate = 115200 Baud / n

n must be selected between 1 and 2304.

Required Baud Rate	Selected Baud Rate
9600	9600
9500	9404
100000	92160
115200	115200

High baud rates shall be used with enabled FIFO, this will avoid loosing data.

EXAMPLE

```
#include “tpmc917.h”

int       fd;
int       status;

/*-------------------------------------------------------------
  Set baud rate of a serial channel specified by fd to 9600
  -------------------------------------------------------------*/
status = ioctl (fd, FIOBAUDRATE, 9600);
if (status != OK)
{
     /* handle Ioctl error */
}
```

ERROR CODES

Error code	Description
S_tp917Drv_IDEV	Invalid device type specified. The device is no serial channel.
S_tp917Drv_IARG	Baudrate out of range

SEE ALSO

ioLib, tyLib, basic I/O routine - *ioctl(),*VxWorks Programmer's Guide: I/O System

4.5.2 FIOFIFO

Select the FIFO trigger levels for receive and transmit FIFO. This Ioctl function is only available for serial channel devices. The special argument **arg** is split into four bytes:

xxxxRRTT

xxxx	the two most significant bytes are unused
RR	the third byte selects the receive trigger [TP917F_NO | TP917F_R8 | TP917F_R16 | TP917F_R56 | TP917F_R60]
TT	the least significant bit selects the transmitter trigger level [TP917F_NO | TP917F_T8 | TP917F_T16 | TP917F_T32 | TP917F_T56]

If one FIFO trigger is set to *TP917F_NO* both FIFOs are disabled. Both FIFO trigger levels must be set up to use the FIFO.

EXAMPLE

```
#include “tpmc917.h”

int       fd;
int       status;

/*-------------------------------------------------------------
  Set FIFO triggering to 8 bytes (receive), 16 bytes (transmit)
  -------------------------------------------------------------*/
status = ioctl (fd, FIOFIFO, (TP917F_R8 << 8) | TP917F_T16);
if (status != OK)
{
     /* handle Ioctl error */
}
```

ERROR CODES

Error code	Description
S_tp917Drv_IDEV	Invalid device type specified. The device is no serial channel.
S_tp917Drv_IARG	Baudrate out of range

SEE ALSO

ioLib, tyLib, basic I/O routine - *ioctl(),*VxWorks Programmer's Guide: I/O System

4.5.3 FIODATABITS

Select the number of data bits in one word - the argument can be set to *[TP917DB_5* | *TP917DB_6* | *TP917DB_7* | *TP917DB_8]* for 5 to 8 data bits. This ioctl function is only available for serial channel devices.

EXAMPLE

```
#include “tpmc917.h”

int       fd;
int       status;

/*-------------------------------------------------------------
  Select datawords with 5 data bits
  -------------------------------------------------------------*/
status = ioctl (fd, FIODATABITS, TP917DB_5);
if (status != OK)
{
     /* handle Ioctl error */
}
```

ERROR CODES

Error code	Description
S_tp917Drv_IDEV	Invalid device type specified. The device is no serial channel.
S_tp917Drv_IARG	Baudrate out of range

SEE ALSO

ioLib, tyLib, basic I/O routine - *ioctl(),*VxWorks Programmer's Guide: I/O System

4.5.4 FIOSTOPBITS

Select the size of the stop bit(s) - allowed values are *TP917SB_10* for one stop bit, *TP917SB_15* or *TP917SB_20* for 1.5 or 2 stop bits. This ioctl function is only available for serial channel devices.

EXAMPLE

```
#include “tpmc917.h”

int       fd;
int       status;

/*-------------------------------------------------------------
  Select 1.5 stop bits
  -------------------------------------------------------------*/
status = ioctl (fd, FIOSTOPBITS, TP917SB_15);
if (status != OK)
{
     /* handle Ioctl error */
}
```

ERROR CODES

Error code	Description
S_tp917Drv_IDEV	Invalid device type specified. The device is no serial channel.
S_tp917Drv_IARG	Baudrate out of range

SEE ALSO

ioLib, tyLib, basic I/O routine - *ioctl(),*VxWorks Programmer's Guide: I/O System

4.5.5 FIOPARITY

Select parity checking - parity checking can be set to even (*TP917EVP*) or odd (*TP917ODP*) parity, or can be disabled (*TP917NOP*). This ioctl function is only available for serial channel devices.

EXAMPLE

```
#include “tpmc917.h”

int       fd;
int       status;

/*-------------------------------------------------------------
  Select parity checking (odd parity)
  -------------------------------------------------------------*/
status = ioctl (fd, FIOPARITY, TP917ODP);
if (status != OK)
{
     /* handle Ioctl error */
}
```

ERROR CODES

Error code	Description
S_tp917Drv_IDEV	Invalid device type specified. The device is no serial channel.
S_tp917Drv_IARG	Baudrate out of range

SEE ALSO

ioLib, tyLib, basic I/O routine - *ioctl(),*VxWorks Programmer's Guide: I/O System

4.5.6 FIOENABLEHWHS

This function enables hardware handshaking. No additional parameter is needed. Hardware handshaking is only allowed when FIFO triggering is enabled. The hardware handshake signals are only generated for controller internal FIFOs, the writing to the VxWorks FIFOs will not be stopped if they are full. So there will be data lost, if the application doesn't read the data fast enough. This Ioctl function is only available for serial channel devices.

EXAMPLE

```
#include "tpmc917.h"

int      fd;
int      status;

/*-------------------------------------------------------------
  Enable hardware handshaking
  ------------------------------------------------------------*/
status = ioctl (fd, FIOENABLEHWHS, 0);
if (status != OK)
{
     /* handle Ioctl error */
}
```

ERROR CODES

Error code	Description
S_tp917Drv_IDEV	Invalid device type specified. The device is no serial channel.

SEE ALSO

ioLib, tyLib, basic I/O routine - *ioctl(),*VxWorks Programmer's Guide: I/O System

4.5.7 FIODISABLEHWHS

This function disables hardware handshaking. No additional parameter is needed.

EXAMPLE

```
#include “tpmc917.h”

int       fd;
int       status;

/*-------------------------------------------------------------
  Disable hardware handshaking
  -------------------------------------------------------------*/
status = ioctl (fd, FIODISABLEHWHS, 0);
if (status != OK)
{
     /* handle Ioctl error */
}
```

ERROR CODES

Error code	Description
S_tp917Drv_IDEV	Invalid device type specified. The device is no serial channel.

SEE ALSO

ioLib, tyLib, basic I/O routine - *ioctl(),*VxWorks Programmer's Guide: I/O System

4.5.8 FIOSETBREAK

Set the break bit of the controller. This will produce a break signal on the transmit line. No additional argument is needed. This ioctl function is only available for serial channel devices.

EXAMPLE

```
#include “tpmc917.h”

int       fd;
int       status;

/*-------------------------------------------------------------
  Set the break flag
  -------------------------------------------------------------*/
status = ioctl (fd, FIOSETBREAK, 0);
if (status != OK)
{
     /* handle Ioctl error */
}
```

ERROR CODES

Error code	Description
S_tp917Drv_IDEV	Invalid device type specified. The device is no serial channel.

SEE ALSO

ioLib, tyLib, basic I/O routine - *ioctl(),*VxWorks Programmer's Guide: I/O System

4.5.9 FIOCLEARBREAK

Remove the break flag of the controller. No additional argument is needed. This ioctl function is only available for serial channel devices.

EXAMPLE

```
#include “tpmc917.h”

int       fd;
int       status;

/*-------------------------------------------------------------
  Clear the break flag
  -------------------------------------------------------------*/
status = ioctl (fd, FIOCLEARBREAK, 0);
if (status != OK)
{
     /* handle Ioctl error */
}
```

ERROR CODES

Error code	Description
S_tp917Drv_IDEV	Invalid device type specified. The device is no serial channel.

SEE ALSO

ioLib, tyLib, basic I/O routine - *ioctl(),*VxWorks Programmer's Guide: I/O System

4.5.10 FIOCHECKBREAK

Check if a break has been received since device creation, reconfiguration or the last *FIOCHECKBREAK* call

A pointer to a char value must be supplied by **arg**, where the result will be returned to. A result of TRUE means that a break has been received. A result of FALSE says that no break has been received. The input break condition will be deleted with this call. This ioctl function is only available for serial channel devices.

EXAMPLE

```
#include “tpmc917.h”

int       fd;
int       status;
char      break_rcv;

/*-------------------------------------------------------------
  Check the break flag
  -------------------------------------------------------------*/
status = ioctl (fd, FIOCHECKBREAK, (int)&break_rcv);
if (status == OK)
{
     if (break_rcv)
     {
          printf("Break signal received.\n");
     } else {
          printf("No Break signal received.\n");
     }
} else {
     /* handle Ioctl error */
}
```

ERROR CODES

Error code	Description
S_tp917Drv_IDEV	Invalid device type specified. The device is no serial channel.

SEE ALSO

ioLib, tyLib, basic I/O routine - *ioctl(),*VxWorks Programmer's Guide: I/O System

4.5.11 FIOCHECKERRORS

Check if errors were detected since device creation, reconfiguration or the last *FIOCHECKERRORS* call. This call needs the pointer to a char value, where the result will be returned to. The result is a flag field with bits set for an error condition. This Ioctl function is only available for serial channel devices.

Bit	Value	Error
0	(1 << 0)	framing error
1	(1 << 1)	parity error

EXAMPLE

```
#include “tpmc917.h”

int       fd;
int       status;
char      errors_rcv;

/*-------------------------------------------------------------
  Read the error flags
  -------------------------------------------------------------*/
status = ioctl (fd, FIOCHECKERRORS, (int)&errors_rcv);
if (status == OK)
{
     if (errors_rcv & (1<<0)) printf("framing error\n");
     if (errors_rcv & (1<<1)) printf("parity error\n");
} else {
     /* handle Ioctl error */
}
```

ERROR CODES

Error code	Description
S_tp917Drv_IDEV	Invalid device type specified. The device is no serial channel.

SEE ALSO

ioLib, tyLib, basic I/O routine - *ioctl(),*VxWorks Programmer's Guide: I/O System

4.5.12 FIORECONFIGURE

This function reconfigures the selected channel. The driver's internal settings will be set to the default configuration and the channel will be set up with its default settings. No additional argument is needed. This Ioctl function is only available for serial channel devices.

EXAMPLE

```
#include “tpmc917.h”

int       fd;
int       status;

/*-------------------------------------------------------------
  Reset the channel to its default values
  -------------------------------------------------------------*/
status = ioctl (fd, FIORECONFIGURE, 0);
if (status != OK)
{
     /* handle Ioctl error */
}
```

ERROR CODES

Error code	Description
S_tp917Drv_IDEV	Invalid device type specified. The device is no serial channel.

SEE ALSO

ioLib, tyLib, basic I/O routine - *ioctl(),*VxWorks Programmer's Guide: I/O System

4.5.13 FIOGETBATSTAT

This function checks the battery state.

If the battery is OK, the function will return without an error. Otherwise, it will return ERROR to indicate the battery's low state. This function is only available for SRAM devices.

EXAMPLE

```
#include “tpmc917.h”

int       fd;
int       status;

/*-------------------------------------------------------------
  Get state of the battery monitors
  -------------------------------------------------------------*/
status = ioctl (fd, FIOGETBATSTAT, 0);
if (status == ERROR)
{
     if ( errnoGet() == S_tp917Drv_BATTLOW)
     {
          printf(“BATTERY WARNING!!!!\n”);
     } else {
          /* handle Ioctl error */
     }
} else {
     printf(“Battery is OK\n”);
}
```

ERROR CODES

Error code	Description
S_tp917Drv_IDEV	Invalid device type specified. The device is no SRAM device.

4.5.14 FIOWAITBATLOW

This function waits until the battery state turns to low, or the specified timeout happens.

A value must be supplied by **arg** to this function which specifies the amount of time to wait for the BatteryLow event. If NO_WAIT is specified, the function returns immediately. If WAIT_FOREVER is specified, the function will wait indefinitely for this event. This function is only available for SRAM devices.

EXAMPLE

```
#include “tpmc917.h”

int       fd;
int       status;

/*-------------------------------------------------------------
  Wait about 100 ticks for battery low event
  -------------------------------------------------------------*/
status = ioctl (fd, FIOWAITBATLOW, 100);
if (status == OK)
{
     printf(“Event occurred, BATTERY WARNING!!!!\n”);
} else {
     if (errnoGet() == S_tp917Drv_TIME)
     {
          printf(“Battery is OK\n”);
     } else {
          /* handle Ioctl error */
     }
}
```

ERROR CODES

Error code	Description
S_tp917Drv_IDEV	Invalid device type specified. The device is no SRAM device.
S_tp917Drv_TIME	The timeout happened, the battery is still OK.

4.5.15 FIOSRAMREAD

This function reads a variable sized data buffer from the SRAM. This function is only available for SRAM devices.

A pointer to a TP917RAM_MEMIO_BUF structure must be passed to this function using **arg**.

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

offset

Specifies the offset relative to the beginning of the SRAM, where the data is to be read. 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 “tpmc917.h”

int                fd;
int                status;
int                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;

…
```

…

```
status = ioctl (fd, FIOSRAMREAD, (int)pMemIoBuf);
if (status == OK)
{
     /* work with data buffer */
} else {
     /* handle Ioctl error */
}
free( pMemIoBuf );
```

ERROR CODES

Error code	Description
S_tp917Drv_IDEV	Invalid device type specified. The device is no SRAM device.
S_tp917Drv_IARG	Invalid argument. Offset + size too big for memory.

4.5.16 FIOSRAMWRITE

This function writes a variable sized data buffer to the SRAM. This function is only available for SRAM devices.

A pointer to a TP917RAM_MEMIO_BUF structure must be passed to this function using **arg**.

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

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 write. 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 “tpmc917.h”

int                fd;
int                status;
int                totalSize;
TP917RAM_MEMIO_BUF *pMemIoBuf;

/*-------------------------------------------------------------
  allocate enough memory and write some bytes to SRAM, Offset=0
  -------------------------------------------------------------*/
totalSize = sizeof(TP917RAM_MEMIO_BUF) + 20*sizeof(unsigned char);
pMemIoBuf = (TP917RAM_MEMIO_BUF*)malloc( totalSize );
memset( pMemIoBuf, 0, totalSize );
sprintf(pMemIoBuf->pData, "Hello World!");

pMemIoBuf->size    = strlen(pMemIoBuf-pData);
pMemIoBuf->offset  = 0;

…
```

…

```
status = ioctl (fd, FIOSRAMWRITE, (int)pMemIoBuf);
if (status != OK)
{
     /* handle Ioctl error */
}
free( pMemIoBuf );
```

ERROR CODES

Error code	Description
S_tp917Drv_IDEV	Invalid device type specified. The device is no SRAM device.
S_tp917Drv_IARG	Invalid argument. Offset + size is too big for memory.

4.5.17 FIOSRAMGETSIZE

This function returns the size of the onboard SRAM in bytes.

A pointer to an unsigned long value must be passed to this function using **arg**.

This function is only available for SRAM devices.

EXAMPLE

```
#include “tpmc917.h”

int           fd;
int           status;
unsigned long SramSize;

/*-------------------------------------------------------------
  Get size of onboard SRAM
  -------------------------------------------------------------*/
status = ioctl (fd, FIOSRAMGETSIZE, (int)&SramSize);
if (status == OK)
{
     printf(“Available SRAM: %d bytes\n”, SramSize);
} else {
     /* handle Ioctl error */
}
```

ERROR CODES

Error code	Description
S_tp917Drv_IDEV	Invalid device type specified. The device is no SRAM device.

4.5.18 FIOSRAMGETPTR

This function returns a pointer to the onboard SRAM for direct access.

A pointer to an unsigned char pointer must be passed to this function using **arg**.

This function is only available for SRAM devices.

EXAMPLE

```
#include <stdio.h>
#include <tyLib.h>
#include <ioLib.h>
#include “tpmc917.h”

int           fd;
int           status;
unsigned char *pSram;

/*-------------------------------------------------------------
  Get pointer to onboard SRAM
  -------------------------------------------------------------*/
status = ioctl (fd, FIOSRAMGETPTR, (int)&pSram);
if (status == OK)
{
     /* write some text to SRAM */
     sprintf( pSram, "Hello World!" );
} else {
     /* handle Ioctl error */
}
```

ERROR CODES

Error code	Description
S_tp917Drv_IDEV	Invalid device type specified. The device is no SRAM device.

5 Appendix

5.1 Predefined Symbols

FIFO Trigger Level

TP917F_NO	4	Disable FIFO (only active if transmit and receive are set to this value)
TP917F_T8	0	Transmit FIFO trigger is 8 byte left in FIFO
TP917F_T16	1	Transmit FIFO trigger is 16 byte left in FIFO
TP917F_T32	2	Transmit FIFO trigger is 32 byte left in FIFO
TP917F_T56	3	Transmit FIFO trigger is 56 byte left in FIFO
TP917F_R8	0	FIFO trigger if 8 received bytes are in FIFO
TP917F_R16	1	FIFO trigger if 16 received bytes are in FIFO
TP917F_R56	2	FIFO trigger if 56 received bytes are in FIFO
TP917F_R60	3	FIFO trigger if 60 received bytes are in FIFO

DATABIT Settings

TP917DB_5	0	Select 5 data bits
TP917DB_6	1	Select 6 data bits
TP917DB_7	2	Select 7 data bits
TP917DB_8	3	Select 8 data bits

STOPBIT Settings

TP917SB_10	0	Select 1 stop bit
TP917SB_15	1	Select 1 to 1.5 stop bits (only if 5 data bits is selected)
TP917SB_20	1	Select 2 stop bits (only if 6, 7 or 8 data bits is selected)

PARITY Settings:

TP917NOP	0	No parity checking
TP917ODP	1	Create and check odd parity
TP917EVP	2	Create and check even parity

ERRORSTATUS Bits:

TP917_FRAMING_ERR	(1 << 0)	Framing error
TP917_PARITY_ERR	(1 << 1)	Parity error

5.2 Additional 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().*

S_tp917Drv_IARG	0x09170100	Invalid argument
S_tp917Drv_BATTLOW	0x09170101	Battery status indicates battery is low
S_tp917Drv_IDEV	0x09170102	Invalid device type
S_tp917Drv_TIME	0x09170103	Timeout during event wait