The Embedded I/O Company

TEWS TECHNOLOGIES

TPCI868-SW-42

VxWorks Device Driver

16 Channel Serial PCI Module

Version 1.0

User Manual

Issue 1.1

April 2004

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

TEWS TECHNOLOGIES LLC
1 E. Liberty Street, Sixth Floor
Reno, Nevada 89504 / USA
Phone: +1 (775) 686 6077
Fax: +1 (775) 686 6024
e-mail: usasales@tews.com
www.tews.com

TPCI868-SW-42

16 Channel Serial PCI Module

VxWorks Device Driver

Issue	Description	Date
1.0	First Issue	April 2001
1.1	General Revision	April 2004

Table of Content

1 Introduction

The TPCI868-SW-42 VxWorks device driver allows the operation of the TPCI868 serial PCI module conforming to the VxWorks I/O system specification. This includes a device-independent basic I/O interface with *open()*, *read()*, *write()* and *ioctl()* functions and a buffered I/O interface (*fopen()*, *printf()*, *scanf()*, ...).

The TPCI868 driver includes the following functions supported by the *VxWorks tty driver support library*:

- 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

Additional the following functions are supported:

- 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, only TPCI868-10)

2 Installation

The software is delivered on a 3½" HD diskette.

Following files are located on the diskette:

tp868drv.c	TPCI868 Driver Source
tpci868.h	TPCI868 Application Include File
tp868def.h	TPCI868 Driver Include File
tp868tst.c	TPCI868 Test and Example Application
tp868pci.c	TPCI868 PCI MMU mapping for Intel x86 based targets
tpxxxhwdep.c	Collection of hardware dependant functions
tpxxxhwdep.h	Include for hardware dependant functions

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

2.1 Hardware dependent Configuration

The driver is written to be used on TEWS carrier boards and others, but sometimes it is necessary to adapt the software. The following points have to be guaranteed:

- full access to the PCI I/O area of the card
- interrupt must be connected

The driver and example support the Motorola MVME2x00 system and generic Intel Pentium system by default.

2.2 Including the driver in VxWorks

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

2.3 Special installation for Intel x86 based targets

The TPCI868 device driver is fully adapted for Intel x86 based targets. This is done by conditional compilation directives inside the source code and controlled by the VxWorks global defined macro ***CPU***. If the contents of this macro is equal to *I80386*, *I80386* or *PENTIUM* special Intel x86 conforming code and function calls will be included.

The second problem for Intel x86 based platforms can't be solved by conditional compilation directives. Due to the fact that some Intel x86 BSP's doesn't map PCI memory spaces of devices which are not used by the BSP, the required PCI memory spaces can't be accessed.

To solve this problem a MMU mapping entry has to be added for the required TPCI868 PCI memory spaces prior the MMU initialization (*usrMmuInit()*) is done.

Please examine the BSP documentation or contact the BSP Vendor whether the BSP perform automatic PCI and MMU configuration or not. If the PCI and MMU initialization is done by the BSP the function *tp868PciInit()* won't be included and the user can skip to the following steps.

The C source file **tp868pci.c** contains the function *tp868PciInit().* This routine finds out all TPCI868 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()*).

If the Tornado 2.0 project facility is used, the right place to call the function *tp868PciInit()* is at the end of the function *sysHwInit()* in **sysLib.c** (can be opened from the project *Files* window).

If Tornado 1.0.1 compatibility tools are used insert the call to *tp868PciInit()* at the beginning of the root task (*usrRoot()*) in **usrConfig.c**.

Be sure that the function is called prior to MMU initialization otherwise the TPCI868 PCI spaces remains unmapped and an access fault occurs during driver initialization.

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

```
tp868PciInit();
```

To link the driver object modules to VxWorks, simply add all necessary driver files to the project. If Tornado 1.0.1 *Standard BSP Builds* is used*...* add the object modules to the macro *MACH_EXTRA* inside the BSP Makefile (MACH_EXTRA = tp868drv.o tp868pci.o ...).

The function *tp868PciInit()* is designed for and tested on generic Pentium targets. If another BSP is used, please refer to BSP documentation or contact the technical support for required adaptation.

If strange errors appeared after system startup with the new build system please carrying out a VxWorks *build clean* and *build all*.

3 I/O system functions

This chapter describes the driver-level interface to the I/O system. The purpose of these functions is to install the driver in the I/O system, add and initialize devices.

3.1 tp868Drv()

NAME

tp868Drv() - installs the TPCI868 driver in the I/O system.

SYNOPSIS

```
STATUS tp868Drv
(
    int                 num_mods,
    TP868_CONF          *confs
)
```

DESCRIPTION

This function installs the TPCI868 driver in the I/O system, sets up interrupt vectors, and performs hardware initialization of the serial ports.

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

PARAMETER

The parameter **num_mods** specifies the number of TPCI868 modules, the driver should use.

The pointer **confs** points to an array of *TP868_CONF* entries. There must be an array entry for each specified module.

The structure has the following layout:

data structure *TP868_CONF:*

```
typedef struct
{
        unsigned int            ModuleAddress;
        unsigned int            IntrAddress;
        unsigned int            ConfAddress;
        int                     vector;
        int                     level;
        unsigned int            ModuleType;
} TP868_CONF;
```

ModuleAddress:	specify the starting address of the register space
IntrAddress:	specify the address of the interrupt status register space
ConfAddress:	specify the starting address local memory configuration space
vector:	specify the interrupt vector (HW dependent)
level:	specify the interrupt level (HW dependent)
ModuleType:	specify the module type TPCI868-10/-11

Most values can be calculated based on values of the PCI Configuration Header (see example *tp868tst.c*).

EXAMPLE

```
#include "tpci868.h"

...

TP868_CONF    tp868conf =
{
     0xfd000100,    /* register space starts at 0xfd000100    */
     0xfd000200,    /* register space starts at 0xfd000200    */
     0xfd000000,    /* register space starts at 0xfd000000    */
     0xF,           /* interrupt vector     */
     0xF,           /* interrupt level      */
     10             /* TPCI868-10 is mounted */
};

...

/*------------------------------------------------------
  Initialize driver for one module at address 0xfe000000
  ------------------------------------------------------*/
status = tp868Drv (1, &tp868conf);

...
```

RETURNS

OK or ERROR (if the driver cannot be installed)

SEE ALSO

VxWorks Programmer's Guide: I/O System

3.2 tp868DevCreate()

NAME

tp868DevCreate() - adds TPCI868 device to the system and initializes device hardware with the selected configuration

SYNOPSIS

```
STATUS tp868DevCreate
(
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            */
TP868_CHANCONF  *devconf    /* device configuration                  */
)
```

DESCRIPTION

This routine creates a device on a specified serial channel that will be serviced by the TPCI868 driver. This function must be called before performing any I/O request to this device.

PARAMETER

The extra parameter **devconf** points to a configuration structure. The configuration structure is compatible to the default structure in the *TP868CNF.H* file. If the pointer is set to zero when calling this function, the default parameter specified in the *TP868CNF.H* file 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;
} TP868_CHANCONF;
```

Baudrate:	Select the initial baud rate of the channel - allowed values are (50 .. 921600)
Options:	Select the initial VxWorks driver options
FifoTrigRcv:	Select the receiver FIFO trigger level (TPCI868-10/-11: use the predefined values in *tpci868.h*) If one of the FIFO trigger levels is set to *TP868F_NO*, the other level is also disabled.
FifoTrigTrm:	Select the transmitter FIFO trigger level (TPCI868-10/-11: use the predefined values in *tpci868.h*) If one of the FIFO trigger levels is set to *TP868F_NO*, the other level is also disabled.
Databits:	Select the number of data bits in one data word Use the predefined values in *tpci868.h*
Stopbits:	Selects the number of stop bits in one data word Use the predefined values in *tpci868.h*
Parity:	Select the parity checking mode - use the predefined values in *tpci868.h*
Handshake:	If hardware handshake is enabled or disabled (after startup) select YES to use the handshake lines, otherwise NO. This parameter is ignored for TPCI868-11.

See also chapter “ioctl()”.

EXAMPLE

```
#include "tpci868.h"

...

TP868_CHANCONF     tp868setup =
{
     115200,                                  /* 115200 Baud    */
     OPT_TERMINAL,                            /* Terminal options     */
     TP868F_NO, TP868F_NO,                    /* no FIFO  */
     TP868DB_8, TP868SB_10, TP868NOP,         /* 8/1/0 Data/Stop/Pari */
     NO                                       /* no handshake   */
};

...

/*-------------------------------------------------------------
  Create the device "/tty1" 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 = tp868DevCreate ("/tty1", 0, 512, 512, &tp868setup);

...
```

RETURNS

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

4 I/O interface functions

This chapter describes the interface to the basic I/O system.

4.1 open()

NAME

open() - opens a device or file.

SYNOPSIS

```
int open
    (
    const char  *name,      /* name of the device to open                    */
    int         flags,      /* not used for TPCI868 driver, must be '0'      */
    int         mode        /* not used for TPCI868 driver, must be '0'      */
    )
```

DESCRIPTION

Before I/O can be performed to the TPCI868 device, a file descriptor must be opened by invoking the basic I/O function *open()*.

PARAMETER

The parameter **name** selects the device which shall be opened.

The parameters **flags** and **mode** are not used and must be 0.

EXAMPLE

```
...

/*-------------------------------------------------------
  Open the device named "/tty1" for I/O
  -------------------------------------------------------*/
fd = open("/tty1", 0, 0);

...
```

RETURNS

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

SEE ALSO

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

4.2 read()

NAME

read() – reads bytes from the specified TPCI868 device.

SYNOPSIS

```
int read
    (
    int         fd,         /* device descriptor from opened TPCI868 device   */
    char        *buffer,    /* pointer to a buffer to receive bytes           */
    size_t      maxbytes    /* maximum number of bytes to read                */
    )
```

DESCRIPTION

This routine reads a number of bytes from the specified serial channel and places them in **buffer**. The parameter **maxbytes** specifies the maximum number of bytes to read.

EXAMPLE

```
int  fd;
int  nbytes;
char buffer[80];

...

/*-------------------------------------------------------------
  Read up to 80 bytes from the serial channel connected with
  the device descriptor fd
  -------------------------------------------------------------*/
nbytes = read (fd, buffer, 80);

...
```

RETURNS

ERROR or number of bytes read

SEE ALSO

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

4.3 write()

NAME

write() – writes bytes to the specified TPCI868 device.

SYNOPSIS

```
int write
        (
        int         fd,         /* device descriptor on which to write          */
        char        *buffer,    /* buffer containing bytes to be written        */
        size_t      nbytes      /* number of bytes to write                     */
        )
```

PARAMETER

This routine writes **nbytes** bytes from **buffer** to the specified serial channel connected with the device descriptor **fd**.

EXAMPLE

```
int  fd;
int  nbytes;

...

/*-------------------------------------------------------------
  Write 12 bytes to the serial channel connected with the
  device descriptor fd
  -------------------------------------------------------------*/
nbytes = write (fd, "Hello world!", 12);

...
```

RETURNS

ERROR or number of bytes written

SEE ALSO

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

4.4 ioctl()

NAME

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

SYNOPSIS

```
int ioctl
(
  int       fd,         /* device descriptor from opened TPCI868 device    */
  int       function,   /* function code       */
  int       arg         /* optional function dependent argument       */
)
```

DESCRIPTION

Special I/O operation that do not fit to the standard basic I/O calls (read, write) will be performed by calling the *ioctl()* function with a specific function code and an optional function dependent argument.

The TPCI868 device driver uses the standard *tty driver support library tyLib*. For details of supported *ioctl* functions see *VxWorks Reference Manual*: tyLib and *VxWorks Programmer's Guide*: I/O System.

PARAMETER

Function codes and function dependent arguments:

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. The selectable baud rates are calculated with:

Baudrate = 921600 Baud / n

n must be selected between 1 and 65536.

Examples:

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

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

Except the VxWorks standard control functions the TPCI868 driver supports some special functions:

FIOFIFO Select the FIFO trigger levels for receive and transmit FIFO

The special argument **arg** is split into four bytes:

TPCI868-10/-11:

xxxxRRTT

xxxx	the two most significant bytes are unused
RR	the third byte selects the receive trigger [TP868F_NO | TP868F_R8 | TP868F_R16 | TP868F_R56 | TP868F_R60]
TT	the least significant bit selects the transmitter trigger level [TP868F_NO | TP868F_T8 | TP868F_T16 | TP868F_T32 | TP868F_T56]

If one FIFO trigger is set to *T868F_NO* both FIFOs are disabled. Both FIFO trigger levels must be set up to use the FIFO.

FIODATABITS Select the number of data bits in one word - the argument can be set to *[TP868DB_5* | *TP868DB_6* | *TP868DB_7* | *TP868DB_8]* for 5 to 8 data bits

FIOSTOPBITS Select the size of the stop bit(s) - allowed values are *TP868SB_10* for one stop bit, *TP868SB_15* or *TP868SB_20* for 1.5 or 2 stop bits

FIOPARITY Select parity checking - parity checking can be set to even (*TP868EVP*) or odd (*TP868ODP*) parity, or can be disabled (*TP868NOP*)

FIOENABLEHWHS Enable hardware handshaking (need no argument)

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 function is only available for TPCI868-10, used for other types will lead to an error.

FIODISABLEHWHS Disable the hardware handshaking (need no argument)

FIOSETBREAK Set the break bit of the controller (need no argument)

This will produce a break signal on the transmit line.

FIOCLEARBREAK Remove the break flag of the controller (need no argument)

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

This call needs the pointer to a char value, 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.

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 flag field with bits set for an error condition.

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

FIORECONFIGURE Reconfigure the selected channel (need no argument)

The driver internal settings will be set to the default configuration and the channel will be set up with its default settings.

EXAMPLE

```
#include <tylib.h>
#include <iolib.h>
#include “tpci868.h”

int  fd;
int  status;
char break_rcv;
char errors_rcv;

/*-------------------------------------------------------------
  Set baud rate of a serial channel specified by fd to 9600
  -------------------------------------------------------------*/
status = ioctl (fd, FIOBAUDRATE, 9600);

/*-------------------------------------------------------------
  Set FIFO triggering to 8 bytes for receive and 16 bytes
  for transmit
  -------------------------------------------------------------*/
status = ioctl (fd, FIOFIFO, (TP868F_R8 << 8) | TP868F_T16);

/*-------------------------------------------------------------
  Select datawords with 5 data bits
  -------------------------------------------------------------*/
status = ioctl (fd, FIODATABITS, TP868DB_5);

/*-------------------------------------------------------------
  select 1.5 stop bits
  -------------------------------------------------------------*/
status = ioctl (fd, FIOSTOPBITS, TP868SB_15);

/*-------------------------------------------------------------
  Select parity checking (odd parity)
  -------------------------------------------------------------*/
status = ioctl (fd, FIOPARITY, TP868ODP);

/*-------------------------------------------------------------
  Enable hardware handshaking
  -------------------------------------------------------------*/
status = ioctl (fd, FIOENABLEHWHS, 0);

/*-------------------------------------------------------------
  Disable hardware handshaking
  -------------------------------------------------------------*/
status = ioctl (fd, FIODISABLEHWHS, 0);
```

```
/*-------------------------------------------------------------
  Set the break flag
  -------------------------------------------------------------*/
status = ioctl (fd, FIOSETBREAK, 0);

/*-------------------------------------------------------------
  Reset baud rate of a serial channel specified by fd to 9600
  -------------------------------------------------------------*/
status = ioctl (fd, FIOCLEARBREAK, 0);

/*-------------------------------------------------------------
  Set the break flag
  -------------------------------------------------------------*/
status = ioctl (fd, FIOCHECKBREAK, (unsigned long)break_rcv);

/*-------------------------------------------------------------
  Read the error flags
  -------------------------------------------------------------*/
status = ioctl (fd, FIOCHECKERRORS, (unsigned long)errors_rcv);

/*-------------------------------------------------------------
  Reset the channel to its default values
  -------------------------------------------------------------*/
status = ioctl (fd, FIORECONFIGURE, 0);
```

RETURNS

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

INCLUDE FILES

tyLib.h, ioLib.h, tpci868.h

SEE ALSO

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

5 Appendix

This chapter describes the symbols which are defined in the file *tpci868.h*.

5.1 Predefined Symbols

FIFO Trigger Level

TP868F_NO	4	Disable FIFO (only active if transmit and receive are set to this value)
TP868F_T8	0	Transmit FIFO trigger is 8 byte left in FIFO
TP868F_T16	1	Transmit FIFO trigger is 16 byte left in FIFO
TP868F_T32	2	Transmit FIFO trigger is 32 byte left in FIFO
TP868F_T56	3	Transmit FIFO trigger is 56 byte left in FIFO
TP868F_R8	0	FIFO trigger if 8 received bytes are in FIFO
TP868F_R16	1	FIFO trigger if 16 received bytes are in FIFO
TP868F_R56	2	FIFO trigger if 56 received bytes are in FIFO
TP868F_R60	3	FIFO trigger if 60 received bytes are in FIFO

DATABIT Settings

TP868DB_5	0	Select 5 data bits
TP868DB_6	1	Select 6 data bits
TP868DB_7	2	Select 7 data bits
TP868DB_8	3	Select 8 data bits

STOPBIT Settings

TP868SB_10	0	Select 1 stop bit
TP868SB_15	1	Select 1 to 1.5 stop bits (only if 5 data bits are selected)
TP868SB_20	1	Select 2 stop bits (only if 6, 7 or 8 data bits are selected)

PARITY Settings:

TP868NOP	0	No parity checking
TP868ODP	1	Create and check odd parity
TP868EVP	2	Create and check even parity

ERRORSTATUS Bits:

TP868_FRAMING_ERR	(1 << 0)	Framing error
TP868_PARITY_ERR	(1 << 1)	Parity error

5.2 Additional Error Codes

S_tp868Drv_IARG	0x08680000	Invalid argument