The Embedded I/O Company

TEWS TECHNOLOGIES

TCP866-SW-42

VxWorks Device Driver

8 Channel Serial Interface

Version 1.0.x

User Manual

Issue 1.0

May 2004

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

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

TCP866-SW-42

8 Channel Serial Interface

VxWorks Device Driver

Issue	Description	Date
1.0	First Issue	May 10, 2004

Table of Content

1 Introduction

The TCP866-SW-42 VxWorks device driver allows the operation of the TCP866 CompactPCI serial interface 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 TCP866 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 TCP866-10 / -11)

2 Installation

The software is delivered on a 3½" HD diskette.

Following files are located on the diskette:

tp866drv.c	TCP866 Driver Source
TCP866.h	TCP866 Application Include File
tp866def.h	TCP866 Driver Include File
tp866tst.c	TCP866 Test and Example Application
tpxxxhwdep.c	Collection of hardware dependant functions
tpxxxhwdep.h	Include for hardware dependant functions
TCP866-SW-42.pdf	This document

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 different platforms. It was developed on an Intel x86 CompactPCI system. The system and the BSP must support PCI auto-configuration, because the driver searches for attached devices itself. Hardware access is provided via special functions. The TCP866 makes no usage of PCI-memory so there is no need to map it for x86-systems separately.

2.2 Including the driver in VxWorks

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

2.3 Example application

The example application shows the basic driver functionality like reading and writing text lines or configuring a channel. A test-option to automatically test the module is provided as well (note: the channels must be wired in a special way, see *tp866tst.c* for details).

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 tp866Drv()

NAME

tp866Drv() - installs the TCP866 driver in the I/O system.

SYNOPSIS

```
STATUS tp866Drv
(
        void
)
```

DESCRIPTION

This function searches for attached TCP866 modules, installs the TCP866 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

No parameters are necessary because the driver searches for attached TCP866 devices itself.

RETURNS

OK or ERROR (if the driver cannot be installed)

SEE ALSO

VxWorks Programmer's Guide: I/O System

3.2 tp866DevCreate()

NAME

tp866DevCreate() - adds a TCP866 device to the system and initializes device hardware with the selected configuration

SYNOPSIS

```
STATUS tp866DevCreate
(
    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            */
    TP866_CHANCONF  *devconf    /* device configuration                  */
)
```

DESCRIPTION

This routine creates a device on a specified serial channel that will be serviced by the TCP866 driver. This function must be called before performing any I/O request to this device.

PARAMETER

The parameter **devconf** points to a configuration structure describing the desired operation mode. If the pointer is set to zero when calling this function, default parameters 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;
} TP866_CHANCONF;
```

Baudrate:	Select the initial baud rate of the channel - allowed values are (25 .. 921600)
Options:	Select the initial VxWorks driver options
FifoTrigRcv:	Select the receiver FIFO trigger level (use the predefined values in *TCP866.h*) If one of the FIFO trigger levels is set to *TP866F_NO*, the other level is also disabled.
FifoTrigTrm:	Select the transmitter FIFO trigger level (use the predefined values in *TCP866.h*) If one of the FIFO trigger levels is set to *TP866F_NO*, the other level is also disabled.

Databits:	Select the number of data bits in one data word Use the predefined values in *TCP866.h*
Stopbits:	Selects the number of stop bits in one data word Use the predefined values in *TCP866.h*
Parity:	Select the parity checking mode - use the predefined values in *TCP866.h*
Handshake:	If hardware handshake is enabled or disabled (after startup) select YES to use the handshake lines, otherwise NO. (Only supported by TCP866-10/-11)

See also chapter “ioctl()”.

EXAMPLE

```
#include "TCP866.h”

...

TP866_CHANCONF     tp866setup =
{
     115200,                                  /* 115200 Baud    */
     OPT_TERMINAL,                            /* Terminal options     */
     TP866F_NO, TP866F_NO,                    /* no FIFO  */
     TP866DB_8, TP866SB_10, TP866NOP,         /* 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
    Handshale: off
  -------------------------------------------------------------*/
status = tp866DevCreate ("/tty1", 0, 512, 512, &tp866setup);

...
```

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 TCP866 driver, must be '0'     */
    int         mode        /* not used for TCP866 driver, must be '0'     */
    )
```

DESCRIPTION

Before I/O can be performed to the TCP866 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 TCP866 device.

SYNOPSIS

```
int read
    (
    int         fd,         /* device descriptor from opened TCP866 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 TCP866 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 TCP866 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 TCP866 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 TCP866 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:

xxxxRRTT

xxxx	the two most significant bytes are unused
RR	the third byte selects the receive trigger [TP866F_NO | TP866F_R8 | TP866F_R16 | TP866F_R56 | TP866F_R60]
TT	the least significant bit selects the transmitter trigger level [TP866F_NO | TP866F_T8 | TP866F_T16 | TP866F_T32 | TP866F_T56]

If one FIFO trigger is set to *T866F_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 *[TP866DB_5* | *TP866DB_6* | *TP866DB_7* | *TP866DB_8]* for 5 to 8 data bits

FIOSTOPBITS Select the size of the stop bit(s) - allowed values are *TP866SB_10* for one stop bit, *TP866SB_15* or *TP866SB_20* for 1.5 or 2 stop bits

FIOPARITY Select parity checking - parity checking can be set to even (*TP866EVP*) or odd (*TP866ODP*) parity, or can be disabled (*TP866NOP*)

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.

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 "TCP866.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, (TP866F_R8 << 8) | TP866F_T16);

/*-------------------------------------------------------------
  Select datawords with 8 data bits
  -------------------------------------------------------------*/
status = ioctl (fd, FIODATABITS, TP866DB_8);

/*-------------------------------------------------------------
  select 1 stop bits
  -------------------------------------------------------------*/
status = ioctl (fd, FIOSTOPBITS, TP866SB_10);

/*-------------------------------------------------------------
  Select parity checking (odd parity)
  -------------------------------------------------------------*/
status = ioctl (fd, FIOPARITY, TP866ODP);

/*-------------------------------------------------------------
  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);

/*-------------------------------------------------------
  Check 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, TCP866.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 *TCP866.h*.

5.1 Predefined Symbols

FIFO Trigger Level

TP866F_NO	4	Disable FIFO (only active if transmit and receive are set to this value)
TP866F_T8	0	Transmit FIFO trigger is 8 byte left in FIFO
TP866F_T16	1	Transmit FIFO trigger is 16 byte left in FIFO
TP866F_T32	2	Transmit FIFO trigger is 32 byte left in FIFO
TP866F_T56	3	Transmit FIFO trigger is 56 byte left in FIFO
TP866F_R8	0	FIFO trigger if 8 received bytes are in FIFO
TP866F_R16	1	FIFO trigger if 16 received bytes are in FIFO
TP866F_R56	2	FIFO trigger if 56 received bytes are in FIFO
TP866F_R60	3	FIFO trigger if 60 received bytes are in FIFO

DATABIT Settings

TP866DB_5	0	Select 5 data bits
TP866DB_6	1	Select 6 data bits
TP866DB_7	2	Select 7 data bits
TP866DB_8	3	Select 8 data bits

STOPBIT Settings

TP866SB_10	0	select 1 stop bit
TP866SB_15	1	select 1 to 1.5 stop bits (only if 5 data bits is selected)
TP866SB_20	1	select 2 stop bits (only if 6, 7 or 8 data bits is selected)

PARITY Settings:

TP866NOP	0	No parity checking
TP866ODP	1	Create and check odd parity
TP866EVP	2	Create and check even parity

ERRORSTATUS Bits:

TP866_FRAMING_ERR	(1 << 0)	Framing error
TP866_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()* or *printErrno().*

S_tp866Drv_IARG	0x08660100	Invalid argument
S_tp866Drv_IBAD	0x08660101	No software detection, local configuration registers are mapped on base address with bit 7 set.