The Embedded I/O Company

TEWS TECHNOLOGIES

TIP865-SW-42

VxWorks Device Driver

4 Channel Serial IP

Version 2.0.x

User Manual

Issue 2.0.2

June 2008

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

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

TIP865-SW-42

VxWorks Device Driver

4 Channel Serial IP

Supported Modules:
TIP865

Issue	Description	Date
1.0	First Issue	June 1995
1.1	Extended Chapter Configuration	May 1998
1.2	General Revision	November 2003
2.0.0	IPAC CARRIER driver support and new user interface	April 24, 2006
2.0.1	Filelist changed, new address TEWS LLC	October 17, 2006
2.0.2	Carrier Driver description added	June 24, 2008

Table of Contents

1 Introduction

1.1 Device Driver

The TIP865-SW-42 VxWorks device driver software allows the operation of the TIP865 IP 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(),…).

The TIP865 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
- option special characters for shell abort and system restart

Additional the following functions are supported:

- baudrates from 2 up to 230400 BAUD
- use 5..8 bit data words
- use 1, 1.5 or 2 stop bits
- optional even or odd parity
- handling of RTS/CTS and DTR/DSR(DCD) line pairs

The TIP865-SW-42 supports the modules listed below:

TIP865-10	4 channel RS232 serial I/O	(IPAC)
TIP865-11	4 channel TTL serial I/O	(IPAC)
TIP865-20	4 channel RS422 serial I/O	(IPAC)
TIP865-30	4 channel RS485 serial I/O	(IPAC)
TIP865-50	2 channel RS485 serial I/O 2 channel RS422 serial I/O	(IPAC)

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

TIP865 User Manual

TIP865 Engineering Manual and Z85C30 SCC (Serial Controller) Manual

CARRIER-SW-42 IPAC Carrier User Manual

1.2 IPAC Carrier Driver

IndustryPack (IPAC) carrier boards have different implementations of the system to IndustryPack bus bridge logic, different implementations of interrupt and error handling and so on. Also the different byte ordering (big-endian versus little-endian) of CPU boards will cause problems on accessing the IndustryPack I/O and memory spaces.

To simplify the implementation of IPAC device drivers which work with any supported carrier board, TEWS TECHNOLOGIES has designed a so called Carrier Driver that hides all differences of different carrier boards under a well defined interface.

The TEWS TECHNOLOGIES IPAC Carrier Driver CARRIER-SW-42 is part of this TIP865-SW-42 distribution. It is located in directory CARRIER-SW-42 on the corresponding distribution media.

This IPAC Device Driver requires a properly installed IPAC Carrier Driver. Due to the design of the Carrier Driver, it is sufficient to install the IPAC Carrier Driver once, even if multiple IPAC Device Drivers are used.

Please refer to the CARRIER-SW-65 User Manual for a detailed description how to install and setup the CARRIER-SW-42 device driver, and for a description of the TEWS TECHNOLOGIES IPAC Carrier Driver concept.

How to use the carrier driver in the application program is shown in the programming example tip865exa.c.

If the IPAC carrier driver isn’t used for the IPAC driver setup, the application software has to setup carrier board hardware, mapping of device memory and interrupt level setup by itself.

2 Installation

The following files and directories are located on the distribution media:

Directory path 'TIP865-SW-42':

tip865drv.c	TIP865 device driver source
tip865def.h	TIP865 driver include file
tip865.h	TIP865 include file for driver and application
tip865conf.h	TIP865 driver configuration file
tip865exa.c	Example application
include/ipac_carrier.h	Carrier driver interface definitions
TIP865-SW-42-2.0.2.pdf	PDF copy of this manual
ChangeLog.txt	Release history
Release.txt	Release information

2.1 Include device driver in Tornado IDE project

For including the TIP865-SW-42 device driver into a Tornado IDE project follow the steps below:

(1) Copy the files from the distribution media into a subdirectory in your project path.
(For example: ./TIP865)

(2) Add the device drivers C-files to your project.
Make a right click to your project in the 'Workspace' window and use the 'Add Files ...' topic.
A file select box appears, and the driver files can be selected.

(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 Tornado User's Guide.

2.2 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	---	---

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.

2.3 Driver Configuration

To adjust the default behavior of the TIP865 device driver see file tip865conf.h and look for the following symbols.

TIP865_MAX_MODULES

This symbol defines the maximum count of supported TIP865 modules. You can increase it to match your system requirements. The default value is 10.

TIP865_DEFAULT_BAUDRATE

This symbol defines the start up speed for all found channels. To modify input and output speed at runtime use FIO_TIP865_BAUDRATE ioctl function. The total default configuration of each channel consists of TIP865_DEFAULT_BAUDRATE, 8 databits, no parity and one stopbit.

TIP865_DEFAULT_OPTIONS

This symbol defines the default VxWorks terminal settings for each serial channel. For more details see also ioLib.h and tyLib.h and look for “OPT_..:” flags.

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 tip865Drv()

NAME

tip865Drv() - installs the TIP865 driver in the I/O system

SYNOPSIS

#include “tip865.h”

STATUS tip865Drv(void)

DESCRIPTION

This function initializes the TIP865 driver and installs it in the I/O system.

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.

EXAMPLE

```
#include "tip865.h”

/*------ Initialize Driver -----*/
status = tip865Drv();
if (status == ERROR)
{
    /* Error handling */
}
```

RETURNS

OK, or ERROR if the function fails.

ERROR CODES

The error codes are stored in *errno* and 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

VxWorks Programmer's Guide: I/O System

3.2 tip865DevCreate()

NAME

tip865DevCreate() – Add a TIP865 serial channel device to the VxWorks system

SYNOPSIS

```
#include "tip865.h"

STATUS tip865DevCreate
(
        char            *name,
        int             devIdx,
        int             funcType,
        void            *pParam
)
```

DESCRIPTION

This routine adds the selected device to the VxWorks system. The device hardware will be setup and prepared for use.

This function must be called before performing any I/O request to this device.

PARAMETER

name

This string specifies the name of the device that will be used to identify the device, for example for *open()* calls.

devIdx

This index number specifies the TIP865 locale serial channel number (0...3) to add to the system. A certain module is described by ipac structure which is part of the *TIP865_DEVCONFIG* buffer (see below). The module descriptor and the locale serial channel number built a unique device index.

If modules of the same type are installed the channel numbers will be advised in the order the IPAC CARRIER *ipFindDevice()* function will find the devices.

Example: (A system with one TIP865-10 on 1st slot and two TIP865-11 on 2nd and 3rd slot) will assign the following global device indexes:

Module	**Device Index**
TIP865-10 (1st to 4th channel)	0..3
TIP865-11 (5th to 8th channel)	4..7
TIP865-11 (9th to 12th channel)	8..11

funcType

This parameter is unused and should be set to *0*.

pParam

This parameter points to a structure (*TIP865_DEVCONFIG*) containing the default configuration of the channel.

The structure (*TIP865_DEVCONFIG*) has the following layout and is defined in tip865.h:

```
typedef struct
{
        struct ipac_resource *ipac;
        int rdBufSize;
        int wrtBufSize;
} TIP865_DEVCONFIG;
```

ipac

Pointer to TIP865 module resource descriptor, retrieved by CARRIER Driver ipFindDevice() function

rdBufSize

Size of input ring buffer in bytes

wrtBufSize

Size of output ring buffer in bytes

EXAMPLE

```
#include  "tip865.h”

…

STATUS             result;
TIP865_DEVCONFIG   tip865Conf;
struct ipac_resource ipac;


… /* IPAC CARRIER Driver initialization */

/*-------------------------------------------------------
```

```
  Create the device "/tip865/0" for the first serial channel
  of the first found module
  -------------------------------------------------------*/
tip865Conf.ipac =  &ipac;
tip865Conf.rdBufSize =  512;
tip865Conf.wrtBufSize = 512;

result = tip865DevCreate(    "/tip865/0",
                             0,
                             0,
                             (void*)&tip865Conf);
if (result == OK)
{
     /* Device successfully created */
}
else
{
     /* Error occurred when creating the device */
}

…
```

RETURNS

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

ERROR CODES

The error codes are stored in *errno* and can be read with the function *errnoGet()*.

Error code	**Description**
S_ioLib_NO_DRIVER	Driver not installed, run tip865Drv()
S_tip865Drv_IARG	Invalid argument in device configuration buffer. Please check all arguments given to tip865DevCreate().
S_ioLib_DEVICE_ERROR	Device error. The certain TIP865 device seems to be faulty or isn't a TIP865 device at all.
S_tip865Drv_IDEV	Device already created.

SEE ALSO

VxWorks Programmer's Guide: I/O System

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 TIP865 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 tip865DevCreate() must be used

flags

Not used

mode

Not used

EXAMPLE

```
int  fd;

…

/*------------------------------------------
  Open the device named "/tip865/0" for I/O
  ------------------------------------------*/
fd = open("/tip865/0", 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 codes are stored in *errno* and 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

```
int 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;
int  retval;

…

/*----------------
  close the device
  ----------------*/
retval = close(fd);
if (retval == 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 codes are stored in *errno* and 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   1024

…

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

…

/*------------------------------------------------------------
  Read data from the TIP865 serial channel connected to “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 codes are stored in *errno* and 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, tyRead, 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 routine writes *nbytes* bytes from *buffer* to the specified serial channel connected with the device descriptor *fd*.

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 write.

EXAMPLE

```
int                fd;
char               buffer[] = “Hello World\n”;
unsigned long      retval;

…

/*-----------------------------------------------------
  Write data to a TIP865 serial channel connected to fd
  ---------------------------------------------------*/
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 codes are stored in *errno* and 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, tyWrite, basic I/O routine - write()

4.5 ioctl()

NAME

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

SYNOPSIS

```
#include "tip865.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 will be performed by calling the *ioctl* function with a specific function code and an optional function dependent argument.

The TIP865 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

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
FIO_TIP865_BAUDRATE	Set baudrate
FIO_TIP865_DATABITS	Set data word length
FIO_TIP865_STOPBITS	Set number of stopbits
FIO_TIP865_PARITY	Set parity checking
FIO_TIP865_CHECKBREAK	Check for pending break
FIO_TIP865_SETBREAK	Set break condition
FIO_TIP865_CLEARBREAK	Clear break condition
FIO_TIP865_RECONFIGURE	Restart channel
FIO_TIP865_CHECKERRORS	Read error status

FIO_TIP865_SETRTS	Set RTS line
FIO_TIP865_CLEARRTS	Clear RTS line
FIO_TIP865_SETDTR	Set DTR line
FIO_TIP865_CLEARDTR	Clear DTR line
FIO_TIP865_CHECKCTS	Retrieve CTS line state
FIO_TIP865_CHECKDSRDCD	Retrieve DSR/DCD line state

arg

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

RETURNS

Function dependent value (described with the function) or ERROR. If the function fails an error code will be stored in *errno*.

ERROR CODES

The error codes are stored in *errno* and 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 - ioctl()

4.5.1 FIO_TIP865_BAUDRATE

This I/O control function sets the baudrate of a certain serial channel. The function specific control parameter **arg** specifies a pointer to an unsigned long value that defines the new baudrate.

The selected baud rate is always set to the next selectable value. The maximum baudrate is 230400 baud except the TIP865-10 (RS232) which is limited to 57600 baud by the transceiver device. If you try to set a higher baudrate than possible for a given transceiver this ioctl function will limit the desired speed to match the certain transceiver specification. In this case the arg pointer will point to a modified baudrate value after ioctl completion.

arg

This parameter points to an unsigned long baudrate value.

Possible baudrates are 2 to 38400, 57600, 115200 and 230400. The range from 2 to 38400 baud is not continues. In the given range you should use baudrates that meet the following formula:

brg = (115200 / baudrate) – 2

with *brg* in the range from 1 to 65535.

For the extended baudrates 57600, 115200 and 230400 the internal baudrate generator is not used.

EXAMPLE

```
#include "tip865.h"
…

int                 fd;
unsigned long       retval;
unsigned long       val;
…

/*------------------------
  Set baudrate to 230400 Baud
  ------------------------*/
val = 230400;

retval = ioctl(fd, FIO_TIP865_BAUDRATE, (int)&val);
if (retval != ERROR){
     /* function succeeded */
     printf("Baudrate set to %d baud.\n", (int)val);
}
else{
     /* handle the error */
}
…
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

The error codes are stored in *errno* and 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).

4.5.2 FIO_TIP865_DATABITS

This I/O control function sets the number of data bits for serial communication. The function specific control parameter **arg** specifies the new configuration

arg

This parameter selects the number of data bits in one word, the argument can be set to [TIP865DB_5 | TIP865DB_6 | TIP865DB_7 | TIP865DB_8] for 5 to 8 data bits.

EXAMPLE

```
#include “tip865.h”
…

int                 fd;
TIP865_ARG          argBuf;
unsigned long       retval;
…

/*----------------------------------------------------
  Execute ioctl() function, set 7 Bit data word length
  ---------------------------------------------------*/

retval = ioctl(fd, FIO_TIP865_DATABITS, TIP865DB_7);
if (retval != ERROR){
     /* function succeeded */
}
else{
     /* handle the error */
}
…
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

The error codes are stored in *errno* and 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) or a driver set code described below. Function specific error codes will be described with the function.

Error code	Description
S_tip865Drv_IARG	Invalid data word length. Check ioctl parameter arg.

4.5.3 FIO_TIP865_STOPBITS

This I/O control function sets the number of stopbits. The function specific control parameter **arg** specifies the new configuration.

arg

This parameter selects the size of the stop bit(s). Allowed values are *TIP865SB_10* for one, *TIP865SB_15* for 1.5 and *TIP865SB_20* for 2 stop bits.

EXAMPLE

```
#include "tip865.h"
…

int                fd;
unsigned long      retval;
…

/*----------------------------------------
  Execute ioctl() function, use one stop bit
  ----------------------------------------*/

retval = ioctl(fd, FIO_TIP865_STOPBITS, TIP865SB_10);
if (retval != ERROR){
     /* function succeeded */
}
else{
     /* handle the error */
}
…
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

The error codes are stored in *errno* and 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) or a driver set code described below. Function specific error codes will be described with the function.

Error code	**Description**
S_tip865Drv_IARG	Invalid stop bits length. Check ioctl parameter arg.

4.5.4 FIO_TIP865_PARITY

This I/O control function sets the parity checking parameters. The function specific control parameter **arg** specifies the new configuration.

arg

This parameter selects parity checking. Parity checking can be set to even (*TIP865EVP*) or odd (*TIP865ODP*) parity, or it can be disabled (*TIP865NOP*).

EXAMPLE

```
#include “tip865.h”
…

int                 fd;
unsigned long       retval;
…

/*----------------------------------------
  Execute ioctl() function, use odd parity
  --------------------------------------*/

retval = ioctl(fd, FIO_TIP865_PARITY, TIP865ODP);
if (retval != ERROR){
     /* function succeeded */
}
else{
     /* handle the error */
}
…
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

The error codes are stored in *errno* and 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) or a driver set code described below. Function specific error codes will be described with the function.

Error code	Description
S_tip865Drv_IARG	Invalid parity checking parameter. Verify ioctl parameter arg.

4.5.5 FIO_TIP865_CHECKBREAK

This I/O control function looks for break conditions. The function specific control parameter **arg** is a pointer to result buffer.

This function checks, if break has been received since device creation, reconfiguration or the last *FIO_TIP865_CHECKBREAK* call.

arg

This parameter is a pointer to a char value, where the result will be stored 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.

EXAMPLE

```
#include “tip865.h”
…

int                 fd;
char                breakCheck;
unsigned long       retval;
…

/*-----------------------------------------------
  Execute ioctl() function, check break condition
  ----------------------------------------------*/

retval = ioctl(fd, FIO_TIP865_CHECKBREAK, (int)&breakCheck);
if (retval != ERROR){
     /* function succeeded */
}
else{
     /* handle the error */
}
…
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

The error codes are stored in *errno* and 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).

4.5.6 FIO_TIP865_SETBREAK

This I/O control function sets the break bit of the controller. This will produce a break signal on the transmit line. The function specific control parameter **arg** is not used for this function.

EXAMPLE

```
#include “tip865.h”

…

int                 fd;
unsigned long       retval;

…

/*------------------------------------------------------------------
  Execute ioctl() function, create break condition on transmit line
  ----------------------------------------------------------------*/

retval = ioctl(fd, FIO_TIP865_SETBREAK, 0);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}

…
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

The error codes are stored in *errno* and 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).

4.5.7 FIO_TIP865_CLEARBREAK

This I/O control function removes the break flag of the controller. The function specific control parameter **arg** is not used for this function.

EXAMPLE

```
#include “tip865.h”

…

int                 fd;
unsigned long       retval;

…

/*-----------------------------------------------------------------
  Execute ioctl() function, clear break condition on transmit line
  ---------------------------------------------------------------*/

retval = ioctl(fd, FIO_TIP865_CLEARBREAK, 0);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}

…
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

The error codes are stored in *errno* and 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).

4.5.8 FIO_TIP865_RECONFIGURE

This I/O control function reconfigures the selected channel. The driver internal settings will be set to the default configuration and the channel will be set up with its default settings. The function specific control parameter **arg** is not used for this function.

EXAMPLE

```
#include “tip865.h”

…

int                 fd;
unsigned long       retval;

…

/*------------------------------------------------------------
  Execute ioctl() function, load and set default configuration
  ----------------------------------------------------------*/

retval = ioctl(fd, FIO_TIP865_RECONFIGURE, 0);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}

…
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

The error codes are stored in *errno* and 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).

4.5.9 FIO_TIP865_CHECKERRORS

This I/O control function checks if errors were detected since device creation, reconfiguration or the last *FIO_TIP865_CHECKERRORS* 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. The function specific control parameter **arg** is a pointer to a char value, where the result will be returned to.

arg

Result pointer. The following flags are set:

bit	**error**
TIP865_FRAMING_ERR	framing error
TIP865_PARITY_ERR	parity error
TIP865_RX_OVERRUN_ERR	rx fifo overrun error

EXAMPLE

```
#include “tip865.h”

…

int                 fd;
char                errorCheck;
unsigned long       retval;

…

/*----------------------------------------------
  Execute ioctl() function, check receive errors
  --------------------------------------------*/

retval = ioctl(fd, FIO_TIP865_CHECKERRORS, (int)&errorCheck);
if (retval != ERROR)
{
     /* function succeeded */
     if (errorCheck & TIP865_FRAMING_ERR)
     {
          /* handle framing errors */
     }

     if (errorCheck & TIP865_PARITY_ERR)
     {
          /* handle parity errors */
     }
```

```
        if (errorCheck & TIP865_RX_OVERRUN_ERR)
        {
             /* handle rx fifo overrun errors */
        }
   }
   else
   {
        /* handle the error */
   }

   …
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

The error codes are stored in *errno* and 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).

4.5.10 FIO_TIP865_SETRTS

This I/O control function sets the RTS line. The function specific control parameter **arg** is not used for this function.

EXAMPLE

```
#include “tip865.h”
…

int                  fd;
unsigned long        retval;
…

/*-----------------------------------------------------------------
  Execute ioctl() function, set RTS line
  ---------------------------------------------------------------*/

retval = ioctl(fd, FIO_TIP865_SETRTS, 0);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}

…
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

The error codes are stored in *errno* and 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).

4.5.11 FIO_TIP865_CLEARRTS

This I/O control function clears the RTS line. The function specific control parameter **arg** is not used for this function.

EXAMPLE

```
#include “tip865.h”
…

int                 fd;
unsigned long       retval;
…

/*------------------------------------------------------------------
  Execute ioctl() function, clear RTS line
  ----------------------------------------------------------------*/

retval = ioctl(fd, FIO_TIP865_CLEARRTS, 0);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}

…
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

The error codes are stored in *errno* and 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).

4.5.12 FIO_TIP865_SETDTR

This I/O control function sets the DTR line. The function specific control parameter **arg** is not used for this function.

EXAMPLE

```
#include “tip865.h”
…

int                fd;
unsigned long      retval;
…

/*------------------------------------------------------------------
  Execute ioctl() function, set DTR line
  ----------------------------------------------------------------*/

retval = ioctl(fd, FIO_TIP865_SETDTR, 0);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}

…
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

The error codes are stored in *errno* and 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).

4.5.13 FIO_TIP865_CLEARDTR

This I/O control function clears the DTR line. The function specific control parameter **arg** is not used for this function.

EXAMPLE

```
#include “tip865.h”
…

int                fd;
unsigned long      retval;
…

/*------------------------------------------------------------------
  Execute ioctl() function, clear DTR line
  ---------------------------------------------------------------*/

retval = ioctl(fd, FIO_TIP865_CLEARDTR, 0);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}

…
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

The error codes are stored in *errno* and 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).

4.5.14 FIO_TIP865_CHECKCTS

This I/O control function checks the CTS line state. The function specific control parameter **arg** is a pointer to a char value, where the result will be returned to.

arg

Result pointer. A result <> 0 means CTS line set. If result is equal to zero the CTS line is clear.

EXAMPLE

```
#include “tip865.h”
…

int                fd;
char               lineState;
unsigned long      retval;
…

/*--------------------------------------------------
  Execute ioctl() function, check CTS line state
  ------------------------------------------------*/

retval = ioctl(fd, FIO_TIP865_CHECKCTS, (int)&lineState);
if (retval != ERROR)
{
     /* function succeeded */
     if (lineState){ printf(“CTS set\n”);}
     else printf(“CTS clear\n”);
}
else
{
     /* handle the error */
}
…
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

The error codes are stored in *errno* and 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).

4.5.15 FIO_TIP865_CHECKDSRDCD

This I/O control function checks the DSR/DCD line state. The DSR/DCD lines on the IO connector share the same pin on the SCC controller. The jumper block on the module decides which line is routed to the SCC. The function specific control parameter **arg** is a pointer to a char value, where the result will be returned to.

arg

Result pointer. A result <> 0 means DSR/DCD line set. If result is equal to zero the DSR/DCD line is clear.

EXAMPLE

```
#include “tip865.h”
…

int                fd;
char               lineState;
unsigned long      retval;
…

/*-------------------------------------------------
  Execute ioctl() function, check DSR/DCD line state
  -----------------------------------------------*/
retval = ioctl(fd, FIO_TIP865_CHECKDSRDCD, (int)&lineState);
if (retval != ERROR)
{
     /* function succeeded */
     if (lineState){ printf(“DSR/DCD set\n”);}
     else printf(“DSR/DCD clear\n”);
}
else
{
     /* handle the error */
}
…
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

The error codes are stored in *errno* and 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).