The Embedded I/O Company

TEWS TECHNOLOGIES

TIP867-SW-42

VxWorks Device Driver

8 Channel Serial RS485 IP

Version 2.0.x

User Manual

Issue 2.0.1

June 2008

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

TIP867-SW-42

VxWorks Device Driver

8 Channel Serial RS485 IP

Supported Modules:
TIP867

Issue	Description	Date
1.0	First Issue	May 1998
1.1	General Revision	November 2003
2.0.0	IPAC CARRIER driver support and new user interface	October 19, 2006
2.0.1	Carrier Driver description added	June 24, 2008

Table of Contents

1 Introduction

1.1 Device Driver

The TIP867-SW-42 VxWorks device driver software allows the operation of the supported TIP867 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(), fprintf(), fscanf()*,…)..

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 TIP867 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 stripping of the parity bit from 8 bit input
- optional special characters for shell abort and system restart

Additional the following functions are supported:

- baud rates from 2 up to 460800 BAUD
- use 5..8 bit data words
- use 1, 1.5 or 2 stop bits
- optional even or odd parity

The TIP867-SW-42 supports the modules listed below:

TIP867-10	8 channel RS485 Interface	(IPAC)
TIP867-20	8 channel RS485 Interface plus 8 serial RS485 clock I/O lines	(IPAC)

In this document all supported modules and devices will be called TIP867. Specials for a 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.

TIP867 User manual

TIP867 Engineering Manual and Z85230 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 TIP867-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 tip867exa.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

Following files are located on the distribution media:

Directory path 'TIP867-SW-42':

tip867drv.c	TIP867 device driver source
tip867def.h	TIP867 driver include file
tip867.h	TIP867 include file for driver and application
tip867exa.c	Example application
include/tdhal.h	Hardware dependent interface functions and definitions
ipac_carrier.h	Carrier driver interface definitions
TIP867-SW-42-2.0.1.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 TIP867-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: ./TIP867)

(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 Special installation for Intel x86 based targets

The TIP867 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.

2.3 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	---

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.

2.4 Driver Configuration

To adjust the default behavior of the TIP867 device driver, refer to file tip867conf.h and look for the following symbols.

TIP867_MAX_MODULES

This symbol defines the maximum count of supported TIP867 modules. You can increase it to match your system requirements. The default value is 10.

TIP867_DEFAULT_BAUDRATE

This symbol defines the start up speed for all found channels. To modify input and output speed at runtime use the FIO_TIP867_BAUDRATE ioctl function. The total default configuration of each channel consists of TIP867_DEFAULT_BAUDRATE, 8 data bits, no parity and one stop bit.

TIP867_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 tip867Drv()

NAME

tip867Drv() - installs the TIP867 driver in the I/O system

SYNOPSIS

#include “tip867.h”

STATUS tip867Drv(void)

DESCRIPTION

This function initializes the TIP867 driver and installs it 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 "tip867.h”

…

STATUS             result;
…

/*-------------------
  Initialize Driver
  -------------------*/
result = tip867Drv();
if (result == ERROR)
{
     /* Error handling */
}

…
```

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()*.

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 tip867DevCreate()

NAME

tip867DevCreate() – Add a TIP867 device to the VxWorks system

SYNOPSIS

#include “tip867.h”

```
STATUS tip867DevCreate
(
    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 TIP867 local serial channel number (0...7) to add to the system. A certain module is described by *ipac* structure which is part of the *TIP867_DEVCONFIG* buffer (see below). The module descriptor and the local serial channel number build a unique device index.

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

Example: A system with one TIP867-10 on 1st slot and two TIP867-20 on 2nd and 3rd slot will assign the following global device indices:

Module	Device Index
TIP867-10 (1st to 8th channel)	0..7
TIP867-20 (9th to 16th channel)	8..15

TIP867-20 (17th to 24th channel) 16..23

funcType

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

pParam

This parameter points to a structure (*TIP867_DEVCONFIG*) containing the default configuration of the channel.

The structure (*TIP867_DEVCONFIG*) has the following layout and is defined in tip867.h:

```
typedef struct
{
    struct ipac_resource *ipac;
    int rdBufSize;
    int wrtBufSize;
} TIP867_DEVCONFIG;
```

ipac

Pointer to TIP867 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 "tip867.h"

…

STATUS             result;
TIP867_DEVCONFIG   tip867Conf;
struct ipac_resource ipac;


…
/* IPAC CARRIER Driver initialization */
```

```
/*
** Find an IP module with manufacturer id MANUFACTOR_TEWS (0xB3)
** and model number MODEL_TIP903 (see tip903.h). This module uses
** interrupts on INT0 and we need need the 16bit memory space. The
** module need an IACK cycle for interrupt handling.
*/
result = ipFindDevice(MANUFACTOR_TEWS, MODEL_TIP903, 0,
     IPAC_INT0_EN | IPAC_LEVEL_SENS | IPAC_CLK_8MHZ |
     IPAC_MEM_16BIT | IPAC_IACK_CYC,
     &ipac);
if (result == OK)
{
     /*-------------------------------------------------------
       Create the device "/tip867/0" for the first serial channel
       of the first found module
       -------------------------------------------------------*/
     tip867Conf.ipac =  &ipac;
     tip867Conf.rdBufSize =  512;
     tip867Conf.wrtBufSize = 512;

     result = tip867DevCreate(    "/tip867/0",
                                  0,
                                  0,
                                  (void*)&tip867Conf);
     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 tip867Drv()
S_tip867Drv_IARG	Invalid argument in device configuration buffer. Please check all arguments given to tip867DevCreate().
S_ioLib_DEVICE_ERROR	Device error. The certain TIP867 device seems to be faulty or isn't a TIP867 device at all.
S_tip867Drv_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 TIP867 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 tip867DevCreate() must be used

flags

Not used

mode

Not used

EXAMPLE

```
int       fd;

…

/*----------------------------------------
  Open the device named "/tip867/0" for I/O
  ----------------------------------------*/
fd = open("/tip867/0", 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

```
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 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];
int           retval;

…

/*-----------------------------
  Read data from TIP867 device
  -----------------------------*/
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, 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 TIP867 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 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 - write()

4.5 ioctl()

NAME

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

SYNOPSIS

#include “tip867.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 TIP867 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_TIP867_BAUDRATE	Set baud rate
FIO_TIP867_DATABITS	Set data word length
FIO_TIP867_STOPBITS	Set number of stop bits
FIO_TIP867_PARITY	Set parity checking
FIO_TIP867_CHECKBREAK	Check for pending break
FIO_TIP867_SETBREAK	Set break condition
FIO_TIP867_CLEARBREAK	Clear break condition
FIO_TIP867_RECONFIGURE	Restart channel
FIO_TIP867_CHECKERRORS	Read error status

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_TIP867_BAUDRATE

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

The selected baud rate is always set to the next selectable value. The maximum baud rate is 460800 baud. If you try to set a higher baud rate 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 baud rate value after ioctl completion.

arg

This parameter points to an unsigned long baud rate value.

Possible baud rates are 2 to 57600, 115200, 230400 and 460800. The range from 2 to 57600 baud is not continuous. In the given range you should use baud rates that meet the following formula:

brg = (230400 / baudrate) – 2

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

For the extended baud rates 115200, 230400 and 460800 the internal baud rate generator is not used.

EXAMPLE

```
#include “tip867.h”
…

int                fd;
unsigned long      retval;
unsigned long      baudrate;
…

/*------------------------
  Set baudrate to 460800 Baud
  ------------------------*/
baudrate = 460800;

retval = ioctl(fd, FIO_TIP867_BAUDRATE, (int)&baudrate);
if (retval != ERROR){
     /* function succeeded */
     printf(“Baudrate set to %d baud.\n”, baudrate);
}
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_TIP867_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 [TIP867DB_5 | TIP867DB_6 | TIP867DB_7 | TIP867DB_8] for 5 to 8 data bits.

EXAMPLE

```
#include “tip867.h”
…

int                fd;
TIP867_ARG         argBuf;
unsigned long      retval;
…

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

retval = ioctl(fd, FIO_TIP867_DATABITS, TIP867DB_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.

Error code	Description
S_tip867Drv_IARG	Invalid data word length. Check ioctl parameter arg.

4.5.3 FIO_TIP867_STOPBITS

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

arg

This parameter selects the size of the stop bit(s). Allowed values are *TIP867SB_10* for one, *TIP867SB_15* for 1.5 and *TIP867SB_20* for 2 stop bits.

EXAMPLE

```
#include “tip867.h”
…

int                fd;
unsigned long      retval;
…

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

retval = ioctl(fd, FIO_TIP867_STOPBITS, TIP867SB_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.

Error code	Description
S_tip867Drv_IARG	Invalid stop bits length. Check ioctl parameter arg.

4.5.4 FIO_TIP867_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 (*TIP867EVP*) or odd (*TIP867ODP*) parity, or it can be disabled (*TIP867NOP*).

EXAMPLE

```
#include “tip867.h”
…

int                fd;
unsigned long      retval;
…

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

retval = ioctl(fd, FIO_TIP867_PARITY, TIP867ODP);
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.

Error code	Description
S_tip867Drv_IARG	Invalid parity checking parameter. Verify ioctl parameter arg.

4.5.5 FIO_TIP867_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 call to this I/O control function.

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 “tip867.h”
…

int               fd;
char              breakCheck;
unsigned long     retval;
…

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

retval = ioctl(fd, FIO_TIP867_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_TIP867_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 “tip867.h”

…

int                fd;
unsigned long      retval;

…

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

retval = ioctl(fd, FIO_TIP867_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_TIP867_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 “tip868.h”

…

int                fd;
unsigned long      retval;

…

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

retval = ioctl(fd, FIO_TIP867_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_TIP867_RECONFIGURE

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

EXAMPLE

```
#include "tip867.h"

…

int                fd;
unsigned long      retval;

…

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

retval = ioctl(fd, FIO_TIP867_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_TIP867_CHECKERRORS

This I/O control function checks if errors were detected since device creation, reconfiguration or the last call to this I/(O control function. 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
TIP867_FRAMING_ERR	framing error
TIP867_PARITY_ERR	parity error
TIP867_RX_OVERRUN_ERR	rx fifo overrun error

EXAMPLE

```
#include “tip867.h”

…

int               fd;
char              errorCheck;
unsigned long     retval;

…

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

retval = ioctl(fd, FIO_TIP867_CHECKERRORS, (int)&errorCheck);
if (retval != ERROR)
{
     /* function succeeded */
     if (errorCheck & TIP867_FRAMING_ERR)
     {
          /* handle framing errors */
     }

     if (errorCheck & TIP867_PARITY_ERR)
     {
          /* handle parity errors */
     }
```

```
     if (errorCheck & TIP867_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).