The Embedded I/O Company

TEWS TECHNOLOGIES

TIP500-SW-42

VxWorks Device Driver

Optically Isolated 16 Channel 12 Bit ADC

Version 2.1.0

User Manual

Issue 2.1.0

July 2010

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

TIP500-SW-42

VxWorks Device Driver

Optically Isolated 16 Channel 12 Bit ADC

Supported Modules:
TIP500-10
TIP500-11
TIP500-20
TIP500-21

Issue	Description	Date
1.0	First Issue	February 6, 1996
1.1	New installation description	April 17, 1998
1.2	Corrections	January 27, 2003
1.2.1	File list modified, new format of Issue	November 5, 2004
2.0.0	complete revision, carrier support added, new application interface	April 20, 2006
2.0.1	New address TEWS LLC, filelist changed, return value of close corrected	February 15, 2007
2.0.2	Carrier Driver description added	June 23, 2008
2.1.0	SMP Support	July 24, 2010

Table of Contents

1 Introduction

1.1 Device Driver

The TIP500-SW-42 VxWorks device driver software allows the operation of the TIP500 IPAC module conforming to the VxWorks I/O system specification. This includes a device-independent basic I/O interface with *open*(), close() and *ioctl()* functions.

The TIP500-SW-42 device driver supports the following features:

- Read converted input data
- Data correction with factory calibration data
- Read module information
- Support for legacy and VxBus IPAC carrier driver
- SMP Support

The TIP500-SW-42 supports the modules listed below:

TIP500-10	Optically isolated 16 channel 12 bit ADC input voltage range +/-10V, gain 1, 2, 5, 10	IndustryPack® compatible
TIP500-11	Optically isolated 16 channel 12 bit ADC input voltage range +/-10V, gain 1, 2, 4, 8	IndustryPack® compatible
TIP500-20	Optically isolated 16 channel 12 bit ADC input voltage range 0V to +10V, gain 1, 2, 5, 10	IndustryPack® compatible
TIP500-21	Optically isolated 16 channel 12 bit ADC input voltage range 0V to +10V, gain 1, 2, 4, 8	IndustryPack® compatible

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

TIP500 User Manual
TIP500 Engineering 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 TIP500-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-42 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.

2 Installation

Following files are located on the distribution media:

Directory path 'TIP500-SW-42':

tip500drv.c	TIP500 device driver source
tip500def.h	TIP500 driver include file
tip500.h	TIP500 include file for driver and application
tip500exa.c	Example application
include/ipac_carrier.h	Carrier driver interface definitions
TIP500-SW-42-2.1.0.pdf	PDF copy of this manual
Release.txt	Release information
ChangeLog.txt	Release history

2.1 Include device driver in VxWorks projects

For including the TIP500-SW-42 device driver into a VxWorks project (e.g. Tornado IDE or Workbench) follow the steps below:

(1) Copy the files from the distribution media into a subdirectory in your project path. (For example: ./TIP500)

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

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

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

2.2 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	< 1 KB
Semaphores	0	2

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

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

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

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

3 I/O system functions

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

3.1 tip500Drv()

NAME

tip500Drv() - installs the TIP500 driver in the I/O system

SYNOPSIS

#include “tip500.h”

STATUS tip500Drv(void)

DESCRIPTION

This function initializes the TIP500 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 "tip500.h”

STATUS result;

/*-------------------
  Initialize Driver
  -------------------*/
result = tip500Drv();
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 code can be read with the function *errnoGet()*.

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

SEE ALSO

VxWorks Programmer's Guide: I/O System

3.2 tip500DevCreate()

NAME

tip500DevCreate() – Add a TIP500 device to the VxWorks system

SYNOPSIS

```
#include “tip500.h”

STATUS tip500DevCreate
(
        char        *name,
        int         devIdx,
        int         funcType,
        void        *pParam
)
```

DESCRIPTION

This function adds the selected device to the VxWorks system. The device hardware will be allocated 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 desired device instance beginning by 0. This parameter is 0 for the first TIP500 in the system, 1 for the second TIP500 and so forth. The order of TIP500 modules depends on the search order of the IPAC carrier driver.

funcType

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

pParam

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

EXAMPLE

```
#include “tip500.h”

STATUS            result;


/*-------------------------------------------------------
  Create the device "/tip500/0" for the first TIP500 module
  -------------------------------------------------------*/

result = tip500DevCreate("/tip500/0", 0, 0, NULL);

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

Error code	Description
S_ioLib_NO_DRIVER	The driver has not been started.
ENXIO	Desired module instance not found
EINVAL	Invalid input argument
EISCONN	The device has already been created
ENOTSUP	The detected model type is not supported
EIO	Device Initialization failed

SEE ALSO

VxWorks Programmer’s Guide: I/O System

4 Basic I/O Functions

The VxWorks basic I/O interface functions are useable with the TIP500 legacy and VxBus-enabled driver in a uniform manner.

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 TIP500 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 tip500DevCreate() must be used

flags

Not used

mode

Not used

EXAMPLE

```
int      fd;

/*----------------------------------------
  Open the device named "/tip500/0" for I/O
  ----------------------------------------*/

fd = open("/tip500/0", 0, 0);

if (fd == ERROR)
{
     /* Handle error */
}
```

RETURNS

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

ERROR CODES

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

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

SEE ALSO

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

4.2 close()

NAME

close() – close a device or file

SYNOPSIS

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

DESCRIPTION

This function closes opened devices.

PARAMETER

fd

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

EXAMPLE

```
int       fd;
STATUS    retval;

/*----------------
  close the device
  ----------------*/

retval = close(fd);

if (retval == ERROR)
{
     /* Handle error */
}
```

RETURNS

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

ERROR CODES

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

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

SEE ALSO

ioLib, basic I/O routine - close()

4.3 ioctl()

NAME

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

SYNOPSIS

```
#include “tip500.h”

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

DESCRIPTION

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

PARAMETER

fd

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

request

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

Function	Description
TIP500_READ	Execute AD conversion and read converted data
TIP500_INFO	Read module information

arg

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

RETURNS

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

ERROR CODES

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

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

SEE ALSO

ioLib, basic I/O routine - ioctl()

4.3.1 TIP500_READ

This I/O control function starts an AD conversion with the specified parameters and will be blocked by a semaphore until the data acquisition and conversion has completed. In case of the input channel or selected gain has changed since the previous conversion the conversion is delayed by the hardware settling time.

Interlocking access to a specific device will be synchronized by a mutual-exclusion semaphore with priority-inheritance. In case of the device is busy converting data the calling task is blocked for 2 ticks at worst until it gains access to the device or the request times out.

The function specific control parameter *arg* is a pointer on a TIP500_READ_BUFFER.

```
typedef struct
{
        int             channel;
        int             gain;
        unsigned long   flags;
        long            data;
} TIP500_READ_BUFFER;
```

channel

This parameter specifies the ADC channel on the specified module. Allowed values are 1 up to 16 for single-ended interfaces and 1 up to 8 for differential interfaces.

gain

This parameter specifies the gain which shall be used for the conversion. The allowed gain values are depending on the installed module type. TIP500-x0 supports gains = 1, 2, 5, and 10, TIP500-x1 support gains = 1, 2, 4 and 8.

flags

This is an ORed value of the following flags defined in tip500.h:

Flag	Description
TIP500_CORRECTION	The ADC value shall be corrected with the factory stored correction data.
TIP500_DIFF	If this flag is set the channel will use a differential input interface. If this flag is not set, the channel will use a single-ended input interface.

data

The result of the conversion will be returned in this parameter. The range of returned values depends on the module type. Unipolar modules will return values between 0 and 4095, and bipolar modules will return values between -2048 and 2047.

EXAMPLE

```
#include “tip500.h”

int                fd;
TIP500_READ_BUFFER readBuf;
int                retval;


/*-------------------------------------
  Read from channel 1 with a gain of 2
  use differential interface and
  make data correction
  -------------------------------------*/
readBuf.channel    = 1;
readBuf.gain       = 2;
readBuf.flags      = TIP500_CORRECTION | TIP500_DIFF;

retval = ioctl(fd, TIP500_READ, (int)&readBuf);
if (retval != ERROR)
{
     /* function succeeded */
     printf("Input Value: %ld\n", readBuf.data);
}
else
{
     /* handle the error */
}
```

ERROR CODES

Error code	Description
EINVAL	An invalid parameter value has been specified.
EBUSY	The module is already in use
EIO	The conversion failed

4.3.2 TIP500_INFO

This I/O control function returns versatile information about the specified device.

The function specific control parameter *arg* is a pointer on a *TIP500_INFO_BUFFER*.

```
typedef struct
{
        int             modelType;
        long            maxVal;
        int             suppGains[4];
        long            corrGain[4];
        long            corrOffset[4];
} TIP500_INFO_BUFFER;
```

modeltype

This parameter returns the model type of the specified device. A TIP500-10 will return 10, a TIP500-11 will return 11 and so on.

maxVal

This parameter returns the maximum positive data value.

suppGains[]

This array returns the supported gain values.

corrGain[]

This array returns the factory stored gain calibration data. (The value is stored in ¼ LSBs).

corrOffset[]

This array returns the factory stored offset calibration data. (The value is stored in ¼ LSBs).

The correction data is assigned to a special gain by its array index. The assignment is made by the *suppGains[]* array.

EXAMPLE

```
#include “tip500.h”


int                fd;
TIP500_INFO_BUFFER infoBuf;
int                retval;


/*-----------------------
  Read module information
  -----------------------*/
retval = ioctl(fd, TIP500_INFO, (int)&infoBuf);

if (retval != ERROR)
{
     /* function succeeded */
     printf("TIP500-%2d\n", infoBuf.modelType);
}
else
{
     /* handle the error */
}
```

ERROR CODES

No function specific error codes.