The Embedded I/O Company

TEWS TECHNOLOGIES

TIP501-SW-42

VxWorks Device Driver

Optically Isolated 16 Channel 16 Bit ADC

Version 2.1.x

User Manual

Issue 2.1.0

January 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

TIP501-SW-42

VxWorks Device Driver

Optically Isolated 16 Channel 16 Bit ADC

Supported Modules:
TIP501

Issue	Description	Date
1.0	First Issue	April 17, 1998
1.1	Corrections	January 27, 2003
2.0.0	Complete revision, carrier support added, new application interface	April 21, 2006
2.0.1	New Address TEWS LLC, ChangeLog.txt added to file list	December 4, 2006
2.0.2	Chapter "Special Configuration for VxWorks 5.4" removed	February 15, 2007
2.0.3	Carrier Driver description added	June 23, 2008
2.1.0	SMP Support	January 25, 2010

Table of Contents

1 INTRODUCTION 4
  1.1 Device Driver 4
  1.2 IPAC Carrier Driver 5
2 INSTALLATION 6
  2.1 Include the device driver in a VxWorks project 6
  2.2 System resource requirement 6
3 I/O SYSTEM FUNCTIONS 7
  3.1 tip501Drv() 7
  3.2 tip501DevCreate() 9
4 I/O FUNCTIONS 13
  4.1 open() 13
  4.2 close() 15
  4.3 ioctl() 17
    4.3.1 TIP501_READ 19
    4.3.2 TIP501_INFO 21

1 Introduction

1.1 Device Driver

The TIP501-SW-42 VxWorks device driver software allows the operation of the TIP501 IPAC conforming to the VxWorks I/O system specification. This includes a device-independent basic I/O interface with *open*(), close() and *ioctl()* functions.

The TIP501-SW-42 device driver supports the following features:

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

The TIP501-SW-42 supports the modules listed below:

TIP501-10	Optically isolated 16 channel 16 bit ADC input voltage range +/-10V, gain 1, 2, 5, 10	IndustryPack® compatible
TIP501-11	Optically isolated 16 channel 16 bit ADC input voltage range +/-10V, gain 1, 2, 4, 8	IndustryPack® compatible
TIP501-20	Optically isolated 16 channel 16 bit ADC input voltage range 0V to +10V, gain 1, 2, 5, 10	IndustryPack® compatible
TIP501-21	Optically isolated 16 channel 16 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 TIP501 devices it is recommended to read the manuals listed below.

TIP501 User manual

TIP501 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 TIP501-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.

How to use the carrier driver in the application program is shown in the programming example tip501exa.c.

2 Installation

Following files are located on the distribution media:

Directory path 'TIP501-SW-42':

tip501drv.c	TIP501 device driver source
tip501def.h	TIP501 driver include file
tip501.h	TIP501 include file for driver and application
tip501exa.c	Example application
include/ipac_carrier.h	Carrier driver interface definitions
TIP501-SW-42-2.1.0.pdf	PDF copy of this manual
Release.txt	Release information
ChangeLog.txt	Release history

2.1 Include the device driver in a VxWorks project

In order to include the TIP501-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: ./TIP501)

(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	---
Semaphores	0	1

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 tip501Drv()

NAME

tip501Drv() - installs the TIP501 driver in the I/O system

SYNOPSIS

#include “tip501.h”

STATUS tip501Drv(void)

DESCRIPTION

This function initializes the TIP501 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  "tip501.h”

…

/*-------------------
  Initialize Driver
  -------------------*/
status = tip501Drv();
if (status == 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 tip501DevCreate()

NAME

tip501DevCreate() – Add a TIP501 device to the VxWorks system

SYNOPSIS

```
#include "tip501.h"

STATUS tip501DevCreate
(
        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 TIP501 minor device number to add to the system.

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

For TIP501 devices there is only one devIdx per hardware module starting with devIdx = 0.

funcType

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

pParam

This parameter points to a structure (*TIP501_DEVCONFIG*) containing the default configuration of the device.

The structure (*TIP501_DEVCONFIG*) has the following layout and is defined in tip501.h:

```
typedef struct
{
        struct ipac_resource *ipac;
} TIP501_DEVCONFIG;
```

ipac

Pointer to TIP501 module resource descriptor, retrieved by CARRIER Driver ipFindDevice() function

EXAMPLE

```
#include "tip501.h"

…

STATUS                  result;
TIP501_DEVCONFIG        tip501Conf;
struct ipac_resource    ipac;

/* IPAC CARRIER Driver initialization */

…

/*
**   Find an IP module from TEWS TECHNOLOGIES (manufacturer = 0xB3)
**   with model number MODEL_TIP501 (see tip501.h).
**   This module uses both interrupt lines and needs an IACK cycle,
**   we need only the IO space base address for the related driver.
*/
result = ipFindDevice(0xB3, MODEL_TIP501, 0,
     IPAC_INT0_EN | IPAC_INT1_EN | IPAC_LEVEL_SENS | IPAC_IACK_CYC |
     IPAC_CLK_8MHZ,
      &ipac);

…
```

```
…

if (result == OK)
{
     /*-----------------------------------------------------------
     Create the device "/tip501/0"
     -----------------------------------------------------------*/
     tip501Conf.ipac =  &ipac;

     result = tip501DevCreate(    "/tip501/0",
                                  0,
                                  0,
                                  (void*)&tip501Conf);
     if (result == OK)
     {
          /* Device successfully created */
     }
     else
     {
          /* Error occurred when creating the device */
     }
}
else
{
     printf("ERROR: No IP found on supported IP carrier boards\n");
}

…
```

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	The driver has not been started.
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 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 TIP501 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 tip501DevCreate() must be used

flags

Not used

mode

Not used

EXAMPLE

```
int  fd;

…

/*------------------------------------------
  Open the device named "/tip501/0" for I/O
  ------------------------------------------*/
fd = open("/tip501/0", 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;
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 ioctl()

NAME

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

SYNOPSIS

```
#include “tip501.h”

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

DESCRIPTION

Special I/O operation that does 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**
TIP501_READ	Execute AD conversion and read value
TIP501_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

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()

4.3.1 TIP501_READ

This I/O control function starts an AD conversion with the specified parameters. The function will wait for completion of the conversion before reading the data. The driver will wait for settling and conversion time if conversion parameters have changed, otherwise it will just wait for conversion time.

The function specific control parameter **arg** is a pointer on a *TIP501_READ_BUFFER*.

```
typedef struct
{
        int                 channel;
        int                 gain;
        unsigned long       flags;
        long                data;
} TIP501_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. TIP501-x0 supports gain = 1, 2, 5, and 10, TIP501-x1 support gain = 1, 2, 4 and 8.

flags

This is an ORed value of the following flags:

Flag	**Description**
TIP501_CORRECTION	The ADC value shall be corrected with the factory stored correction data.
TIP501_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 65535, and bipolar modules will return values between -32768 and 32767.

EXAMPLE

```
#include “tip501.h”

…

int                fd;
TIP501_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      = TIP501_CORRECTION | TIP501_DIFF;

retval = ioctl(fd, TIP501_READ, (int)&readBuf);
if (retval != ERROR)
{
     /* function succeeded */
     printf("Input Value: %ld\n", readBuf.data);
}
else
{
     /* handle the error */
}

…
```

ERROR CODES

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

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 TIP501_INFO

This I/O control function returns information about the specified device. The function specific control parameter **arg** is a pointer on a *TIP501_INFO_BUFFER*.

```
typedef struct
{
        int             modelType;
        long            maxVal;
        int             suppGains[4];
        long            corrGain[4];
        long            corrOffset[4];
} TIP501_INFO_BUFFER;
```

modeltype

This parameter returns the model type of the specified device. A TIP501-10 will return 10, a TIP501-11 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 “tip501.h”

…

int                fd;
TIP501_INFO_BUFFER infoBuf;
int                retval;

…

/*-----------------------
  Read module information
  -----------------------*/
retval = ioctl(fd, TIP501_INFO, (int)&infoBuf);
if (retval != ERROR)
{
     /* function succeeded */
     printf("TIP501-%2d\n", infoBuf.modelType);
}
else
{
     /* handle the error */
}

…
```

ERROR CODES

No function specific error codes.