The Embedded I/O Company

TEWS TECHNOLOGIES

TIP551-SW-42

VxWorks Device Driver

Optically Isolated 4 Channel 16 Bit D/A

Version 2.1.x

User Manual

Issue 2.1.0

March 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

TIP551-SW-42

VxWorks Device Driver

Optically Isolated 4 Channel 16 Bit D/A

Supported Modules:
TIP551

This document contains information, which is proprietary to TEWS TECHNOLOGIES GmbH. Any reproduction without written permission is forbidden.

TEWS TECHNOLOGIES GmbH has made any effort to ensure that this manual is accurate and complete. However TEWS TECHNOLOGIES GmbH reserves the right to change the product described in this document at any time without notice.

TEWS TECHNOLOGIES GmbH is not liable for any damage arising out of the application or use of the device described herein.

©1997-2010 by TEWS TECHNOLOGIES GmbH

Issue	Description	Date
	Preliminary Issue	August, 1997
1.0	First Issue	September, 1997
1.1	General Update	July, 2002
2.0.0	Complete revision, carrier support added, new application interface	April 25, 2006
2.0.1	New Address TEWS TECHNOLOGIES LLC, ChangeLog.txt added.	January 3, 2007
2.0.2	Carrier Driver description added	June 24, 2008
2.1.0	SMP support	March 3, 2010

Table of Contents

1 Introduction

1.1 Device Driver

The TIP551-SW-42 VxWorks device driver software allows the operation of the TIP551 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 TIP551-SW-42 device driver supports the following features:

- Write data into DAC data register with and without conversion
- Data correction with factory set data
- Read module information
- Support for legacy and VxBus IPAC carrier driver
- SMP Support

The TIP551-SW-42 supports the modules listed below:

TIP551	4 Channel 16 Bit DAC	IndustryPack® compatible

To get more information about the features and use of TIP551 devices it is recommended to read the manuals listed below.

TIP551 User manual
TIP551 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 TIP551-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 tip551exa.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 'TIP551-SW-42':

tip551drv.c	TIP551 device driver source
tip551def.h	TIP551 driver include file
tip551.h	TIP551 include file for driver and application
tip551exa.c	Example application
include/ipac_carrier.h	Carrier driver interface definitions
TIP551-SW-42-2.1.0.pdf	PDF copy of this manual
ChangeLog.txt	Release history
Release.txt	Release information

2.1 Include the device driver in a VxWorks project

In order to include the TIP551-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: ./TIP551)

(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	---
Semaphore	---	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 tip551Drv()

NAME

tip551Drv() - install the TIP551 driver in the I/O system

SYNOPSIS

#include “tip551.h”

STATUS tip551Drv(void)

DESCRIPTION

This function initializes the TIP551 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  "tip551.h”

STATUS    status

/*-------------------
  Initialize Driver
  -------------------*/
status = tip551Drv();
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 tip551DevCreate()

NAME

tip551DevCreate() – Add a TIP551 device to the VxWorks system

SYNOPSIS

```
#include “tip551.h”

STATUS tip551DevCreate
(
        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 TIP551 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 TIP551 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 is unused and should be set to NULL.

EXAMPLE

```
#include  "tip551.h”

STATUS                  result;

/*--------------------------------------------------------
  Create the device "/tip551/0"
  --------------------------------------------------------*/
result = tip551DevCreate(    "/tip551/0",
                             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 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 TIP551 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 tip551DevCreate() must be used

flags

Not used

mode

Not used

EXAMPLE

```
int  fd;

/*------------------------------------------
  Open the device named "/tip551/0" for I/O
  ------------------------------------------*/
fd = open("/tip551/0", 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

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

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() - perform an I/O control function

SYNOPSIS

```
#include "tip551.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**
TIP551_WRITE	Load data value and execute DA conversion
TIP551_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 TIP551_WRITE

This I/O control function loads the specified (or corrected) output value for the specified channel, and if requested starts a DA conversion. The function specific control parameter **arg** is a pointer to a *TIP551_WRITE_BUFFER* structure.

If a conversion is still busy the function will wait for completion.

```
typedef struct
{
        int                 channel;
        unsigned long       flags;
        long                data;
} TIP551_WRITE_BUFFER;
```

channel

This parameter specifies the DAC channel on the specified module the data value shall be loaded to. Allowed values are 1 up to 4.

flags

This is an ORed value of the following flags defined in tip551.h:

Flag	**Description**
TIP551_CORRECTION	The DAC value shall be corrected with the factory stored correction data.
TIP551_NOUPDATE	If this flag is set, the data value will just be loaded to the channel, but no conversion will be started (see also TIP551 User Manual - Channel Select Register). The conversion will be started with the next *TIP551_WRITE* access to a channel of the specified module with this flag not set. If this flag is not set, the data value will be loaded to the channel and the conversion will be started immediately.

data

This parameter specifies the new conversion value. The range of allowed values depends on the selected output range. In unipolar mode (0V … +10V) allowed values are between 0 and 65535 and in bipolar mode (-10V … +10V) allowed values are between -32768 and 32767.

EXAMPLE

```
#include “tip551.h”

int                     fd;
TIP551_WRITE_BUFFER     writeBuf;
int                     retval;

/*-----------------------------------
  Write a value of 0x1000 to channel 3
  make data correction
  -----------------------------------*/
writeBuf.channel   = 3;
writeBuf.flags     = TIP551_CORRECTION;
writeBuf.data      = 0x1000;

retval = ioctl(fd, TIP551_WRITE, (int)&writeBuf);
if (retval == ERROR)
{
     /* handle the error */
}
```

ERROR CODES

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.
EIO	The conversion failed.

4.3.2 TIP551_INFO

This I/O control function returns information about the specified device. The function specific control parameter **arg** is a pointer to a *TIP551_INFO_BUFFER* structure.

```
typedef struct
{
        int             modelType;
        long            selRange;
        long            corrGain[4];
        long            corrOffset[4];
} TIP551_INFO_BUFFER;
```

modelType

This parameter returns the model type of the specified device. A TIP551-10 will return 10.

selRange

This parameter returns the currently selected output range. The following ranges can be returned:

Value	**Description**
TIP551_UNIPOL	The module is switched to unipolar mode. Voltage range is 0V … 10V.
TIP551_BIPOL	The module is switched to bipolar mode. Voltage range is -10V … +10V.

corrGain[]

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

corrOffset[]

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

The correction data is assigned to a special channel by its array index. Index 0 selects correction data of channel 1, Index 1 of channel 2, and so on.

EXAMPLE

```
#include “tip551.h”

int                fd;
TIP551_INFO_BUFFER infoBuf;
int                retval;

/*-----------------------
  Read module information
  -----------------------*/
retval = ioctl(fd, TIP551_INFO, (int)&infoBuf);
if (retval != ERROR)
{
     /* function succeeded */
     printf("TIP551-%2d\n", infoBuf.modelType);
}
else
{
     /* handle the error */
}
```