The Embedded I/O Company

TEWS TECHNOLOGIES

TIP850-SW-42

VxWorks Device Driver

16 Channel 12 bit A/D

4 Channel 12 bit D/A

Version 1.2.x

User Manual

Issue 1.2.2

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

TIP850-SW-42

VxWorks Device Driver

16 Channel 12 bit A/D
4 Channel 12 bit D/A

Supported Modules:
TIP850

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.

©2006-2008 by TEWS TECHNOLOGIES GmbH

Issue	Description	Date
1.0	First Issue	May 1996
1.1	General Revision	November 2003
1.2.0	IPAC Carrier Driver Support	January 11, 2006
1.2.1	New Address TEWS LLC, General Revision	February 06, 2008
1.2.2	Carrier Driver description added	June 24, 2008

Table of Contents

1 INTRODUCTION 4
1.1 Device Driver 4
1.2 IPAC Carrier Driver 5
2 INSTALLATION 6
2.1 Include device driver in Tornado IDE project 6
2.2 System resource requirement 7
3 I/O SYSTEM FUNCTIONS 8
3.1 tip850Drv() 8
3.2 tip850DevCreate() 10
4 I/O FUNCTIONS 13
4.1 open() 13
4.2 close() 15
4.3 ioctl() 17
4.3.1 FIO_TIP850_SETMODE 19
4.3.2 FIO_TIP850_SETGAIN 20
4.3.3 FIO_TIP850_READ_ADC 21
4.3.4 FIO_TIP850_WRITE_DAC 23

1 Introduction

1.1 Device Driver

The TIP850-SW-42 VxWorks device driver software allows the operation of the TIP850 IP conforming to the VxWorks I/O system specification. This includes a device-independent basic I/O interface with open(), close() and ioctl() functions.

The TIP850-SW-42 driver includes the following functions:

- Reading ADC data (optional data correction with factory calibrated data)
- Programming gain for ADC channels
- Programming single or differential ADC Inputs
- Writing DAC data (optional data correction with factory calibrated data)

The TIP850-SW-42 driver supports the modules listed below:

TIP850-10/-11	16 Channel 12 bit A/D 4 Channel 12 bit D/A	(IndustryPack ®)

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

TIP850 User manual

TIP850 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 TIP850-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 tip850exa.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 'TIP850-SW-42':

tip850drv.c	TIP850 device driver source
tip850def.h	TIP850 driver include file
tip850.h	TIP850 include file for driver and application
tip850exa.c	Example application
include/ipac_carrier.h	Carrier driver interface definitions
TIP850-SW-42-1.2.2.pdf	PDF copy of this manual
Release.txt	Release information
ChangeLog.txt	Release history

2.1 Include device driver in Tornado IDE project

For Including the TIP850-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: ./TIP850)

(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	---	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 tip850Drv()

NAME

tip850Drv() - installs the TIP850 driver in the I/O system

SYNOPSIS

#include “tip850.h”

STATUS tip850Drv(void)

DESCRIPTION

This function initializes the TIP850 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 "tip850.h”

/*------ Initialize Driver -----*/
status = tip850Drv();
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 tip850DevCreate()

NAME

tip850DevCreate() – Add a TIP850 device to the VxWorks system

SYNOPSIS

#include “tip850.h”

```
STATUS tip850DevCreate
(
        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 TIP850 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 TIP850 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 (*TIP850_DEVCONFIG*) containing the default configuration of the channel.

The structure (*TIP850_DEVCONFIG*) has the following layout and is defined in tip850.h:

```
typedef struct
{
    struct ipac_resource *ipac;
} TIP850_DEVCONFIG;
```

ipac

Pointer to TIP850 module resource descriptor, retrieved by CARRIER Driver ipFindDevice() function

EXAMPLE

```
#include "tip850.h"

STATUS             result;
TIP850_DEVCONFIG   tip850Conf;
struct ipac_resource ipac;

/* IPAC CARRIER Driver initialization */

/*
**   Find an IP module from TEWS TECHNOLOGIES (manufacturer = 0xB3)
**   with model number MODEL_TIP850 (see tip850.h). This module does not
**   use interrupts and we need only the IO space base address for the
**   related driver.
*/
result = ipFindDevice(0xB3, MODEL_TIP850, 0,
     IPAC_INT0_EN | IPAC_LEVEL_SENS | IPAC_CLK_8MHZ,
      &ipac);

if (result == OK) {
     devConfig.ipac = &ipac;

     /*-------------------------------------------------------
     Create the device "/tip850/0"
     -------------------------------------------------------*/
     tip850Conf.ipac =  &ipac;

…
```

```
…

     result = tip850DevCreate(    "/tip850/0",
                              0,
                              0,
                              (void*)&tip850Conf);
     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	Driver not installed, run tip850Drv()
S_tip850Drv_IARG	Invalid argument in device configuration buffer. Please check all arguments given to tip850DevCreate().
S_ioLib_DEVICE_ERROR	Device error. The certain TIP850 device seems to be faulty or isn't a TIP850 device at all.
EISCONN	The Device has already been 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 TIP850 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 tip850DevCreate() must be used

flags

Not used

mode

Not used

EXAMPLE

```
int  fd;

/*----------------------------------------
  Open the device named "/tip850/0" for I/O
  ----------------------------------------*/
fd = open("/tip850/0", 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 ioctl()

NAME

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

SYNOPSIS

```
#include “tip850.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.

For details of supported *ioctl* functions see VxWorks Reference Manual: 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_TIP850_SETMODE	Set differential or single ended mode AD conversion
FIO_TIP850_SETGAIN	Set gain for AD conversion
FIO_TIP850_READ_ADC	Read from an ADC channel
FIO_TIP850_WRITE_DAC	Write to a DAC channel

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 FIO_TIP850_SETMODE

NAME

FIO_TIP850_SETMODE – Set differential or single ended mode AD conversion

DESCRIPTION

This function is used to define if the ADC input multiplexer shall be configured to 16 single-ended channels or to 8 differential channels.

arg

This parameter specifies the mode for AD conversion. Valid values are:

T850_SINGLE	TIP850 ADC inputs are configured for 16 single-ended channels.
T850_DIFF	TIP850 ADC inputs are configured for 8 differential channels.

EXAMPLE

```
#include “tip850.h”

int  status;
int  fd;

/*-----------------------------------------------------
  Select differential input mode
  -----------------------------------------------------*/
status = ioctl(fd, FIO_TIP850_SETMODE, T850_DIFF);

/*-----------------------------------------------
  Select single-ended input mode
  -----------------------------------------------*/
status = ioctl(fd, FIO_TIP850_SETMODE, T850_SINGLE);
```

SEE ALSO

ioLib, basic I/O routine - ioctl(), VxWorks Programmer's Guide: I/O System.

4.3.2 FIO_TIP850_SETGAIN

NAME

FIO_TIP850_SETGAIN – Set gain for AD conversion

DESCRIPTION

This function is used to define the gain of the programmable amplifier for the ADC inputs. The selected gain (symbols are predefined in *tip850.h*) must be specified in the function dependent argument *arg*.

arg

Specifies the gain, which shall be used to read the AD value. Valid gains are:

T850_GAIN_1	Select Gain 1x	Valid for TIP850-10/-11
T850_GAIN_10	Select Gain 10x	Valid for TIP850-10
T850_GAIN_100	Select Gain 100x	Valid for TIP850-10
T850_GAIN_2	Select Gain 2x	Valid for TIP850-11
T850_GAIN_4	Select Gain 4x	Valid for TIP850-11
T850_GAIN_8	Select Gain 8x	Valid for TIP850-11

SEE ALSO

ioLib, basic I/O routine - ioctl(), VxWorks Programmer's Guide: I/O System.

4.3.3 FIO_TIP850_READ_ADC

NAME

FIO_TIP850_READ_ADC – Read from a ADC channel

DESCRIPTION

The ioctl() function is used to read the value of a specified ADC channel. This function will use the ADC mode and gain which has been programmed with the latest call of the *ioctl* function FIO_TIP850_SETMODE and FIO_TIP850_SETGAIN.

The argument *arg* points to a driver-specific I/O parameter structure of type T850_IO_BUFFER. The data structure *T850_IO_BUFFER* has the following layout:.

```
typedef struct
{
        unsigned int        flags;
        unsigned int        channel;
        int                 data;
        int                 timeout;
} T850_IO_BUFFER;
```

flags

If the ADC channel, specified by the parameter *channel*, has changed from a previous call of *read*, the data conversion will be automatically delayed until the settling time for the programmable gain amplifier has expired to get the best accuracy of the converted value. However this delay can be disabled by setting the flag *T850_IGNORE* in *flags*.

The flag *T850_CORRECTION* in *flags* will enable the correction of the gain and offset error of the TIP850 with the factory calibration information which is stored in the ID PROM.
All errors are considered to be linear. For each of the programmable gains two correction numbers are used. One corrects for the offset (or zero) error, and the second corrects for gain error (see also Hardware User Manual, chapter “Data correction”).

channel

This parameter specifies the channel to convert. For single ended mode valid range is 1..16. For differential mode use range 1..8.

data

If this ioctl() function succeeds this parameter receives the converted ADC channel data.

timeout

This parameter specifies the maximum time in system ticks the user is wiling to wait for ioctl completion. A value of -1 means “wait forever”, a value of 0 means “no wait” and a value greater than 0 means “wait *timeout* system ticks”.

EXAMPLE

```
#include “tip850.h”

T850_IO_BUFFER     IObuf;
int                fd;
STATUS             status;
int                error;

/*-------------------------------------------------------------
  Start a conversion on ADC channel 1 and correct the result
  of the conversion by the factory calibration information
  -------------------------------------------------------------*/
IObuf.flags   = T850_CORRECTION;
IObuf.channel = 1;

status = ioctl (fd, FIO_TIP850_READ_ADC, (int)&IObuf);

if (status == ERROR)
{
     printf(“Device read error errno = %xh\n”, errnoGet());
}
else
{
     printf(“ADC value = %d\n”, IObuf.data);
}
```

ERRORS

S_t850Drv_ICHAN	Invalid ADC channel.

SEE ALSO

ioLib, basic I/O routine - ioctl(), VxWorks Programmer's Guide: I/O System.

4.3.4 FIO_TIP850_WRITE_DAC

NAME

FIO_TIP850_WRITE_DAC – Write to a DAC channel

DESCRIPTION

This ioctl function is used to write the value of a specified DAC channel.

The argument *arg* points to a driver-specific I/O parameter structure of type T850_IO_BUFFER. The data structure *T850_IO_BUFFER* has the following layout:

```
typedef struct
{
        unsigned int    flags;
        unsigned int    channel;
        int             data;
        int             timeout;
} T850_IO_BUFFER;
```

flags

The flag *T850_CORRECTION* in *flags* will enable the correction of the gain and offset error of the TIP850 with the factory calibration information which is stored in the ID PROM.
All errors are considered to be linear. For each DAC channel two correction numbers are used. One corrects for the offset (or zero) error, and the second corrects for gain error (see also Hardware User Manual, chapter “Data correction”).

channel

This parameter specifies the channel to convert. Valid channels are 1..4.

data

This parameter specifies the value to be send to the certain DAC channel.

timeout

This parameter is not used for this function.

EXAMPLE

```
#include “tip850.h”

T850_IO_BUFFER     IObuf;
int                fd, error;
STATUS             status;

/*-------------------------------------------------------------
  Start a conversion on DAC channel 1 after correcting the
  DAC value by the factory calibration information
  DAC channel 1 shall be set to -5V
  -------------------------------------------------------------*/
IObuf.flags   = T850_CORRECTION;
IObuf.channel = 1;
IObuf.data    = -1024;

status = ioctl (fd, FIO_TIP850_WRITE_DAC, (int)&IObuf);

if (status == ERROR)
{
     printf(“Device write error errno = %xh\n”, errnoGet());
}
else
{
     printf(“OK – DAC1 set to -5V\n”);
}
```

ERRORS

S_t850Drv_ICHAN	Invalid DAC channel.

SEE ALSO

ioLib, basic I/O routine - ioctl(), VxWorks Programmer's Guide: I/O System.