The Embedded I/O Company

TEWS TECHNOLOGIES

TIP830-SW-42

VxWorks Device Driver

8 Channel Simultaneous Sampling ADC

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

TIP830-SW-42

VxWorks Device Driver

8 Channel Simultaneous Sampling ADC

Supported Modules:
TIP830-10
TIP830-20

Issue	Description	Date
1.0	First Issue	June 1996
1.1	General Revision	August 2003
1.1.1	New file list	June 15, 2005
2.0.0	Carrier Driver Support added, General Revision of this Manual	August 15, 2007
2.0.1	Carrier Driver description added	June 24, 2008

Table of Contents

1 Introduction

1.1 Device Driver

The TIP830-SW-42 VxWorks device driver software allows the operation of the supported IndustryPack conforming to the VxWorks I/O system specification. This includes a device-independent basic I/O interface with *open*(), *close(),* and *ioctl()* functions.

The TIP830-SW-42 device driver supports the following features:

- Read all channels
- Read all channels and correct input data

The TIP830-SW-42 supports the modules listed below:

TIP830-10	8 Channel synchronous 12 Bit ADC	IndustryPack
TIP830-20	8 Channel synchronous 16 Bit ADC	IndustryPack

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

TIP830 User manual

TIP830 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 TIP830-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 tip830exa.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 'TIP830-SW-42':

tip830drv.c	TIP830 device driver source
tip830def.h	TIP830 driver include file
tip830.h	TIP830 include file for driver and application
tip830exa.c	Example application
include/ipac_carrier.h	Carrier driver interface definitions
TIP830-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 TIP830-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: ./TIP830)

(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 TIP830 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.

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 tip830Drv()

NAME

tip830Drv() - installs the TIP830 driver in the I/O system

SYNOPSIS

#include “tip830.h”

STATUS tip830Drv(void)

DESCRIPTION

This function installs the TIP830 driver 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 "tip830.h”

STATUS            result;

/*-------------------
  Initialize Driver
  -------------------*/
result = tip830Drv();
if (result == ERROR)
{
     /* Error handling */
}
```

RETURNS

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

ERROR CODES

Error codes are only set by system functions. The error codes are stored in *errno* and can be read with the function *errnoGet()*.

SEE ALSO

VxWorks Programmer's Guide: I/O System

3.2 tip830DevCreate()

NAME

tip830DevCreate() – Add a TIP830 device to the VxWorks system

SYNOPSIS

```
#include “tip830.h”

STATUS tip830DevCreate
(
        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 device to add to the system.

funcType

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

pParam

This parameter points to a structure (*TIP830_DEVCONFIG*) containing the default configuration of the device.

typedef struct
{
    struct ipac_resource *ipac;
} TIP830_DEVCONFIG;

ipac

Pointer to TIP836 module resource descriptor, retrieved by CARRIER Driver ipFindDevice() function

EXAMPLE

```
#include "tip830.h"

STATUS                  result;
TIP830_DEVCONFIG        tip830Conf;
struct ipac_resource    ipac;

/* IPAC CARRIER Driver initialization */
…
/*
**   Find an IP module from TEWS TECHNOLOGIES (MANUFACTOR_TEWS)
**   with model number MODEL_TIP830 (see tip830.h). This module uses
**   no interrupt line and we need only the IO space base address for the
**   related driver.
*/
result = ipFindDevice(MANUFACTOR_TEWS, MODEL_TIP830_10, 0,
     IPAC_CLK_8MHZ,
     &ipac);

if (result == OK) {
     tip830Conf.ipac = &ipac;

     /*-------------------------------------------------------
     Create the device "/tip830_0"
     -------------------------------------------------------*/
     tip830Conf.ipac =  &ipac;

…
```

```
…

     result = tip830DevCreate(    "/tip830_0",
                                  0,
                                  0,
                                  (void*)&tip830Conf);
     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	Input parameter has an invalid value
EBUSY	The device has already been created
ENOTSUP	Invalid model type detected
ETIMEDOUT	An initial ADC conversion timed out

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 TIP830 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 tip830DevCreate() must be used

flags

Not used

mode

Not used

EXAMPLE

```
int      fd;

/*----------------------------------------
  Open the device named "/tip830_0" for I/O
  ----------------------------------------*/
fd = open("/tip830_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 “tip830.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
FIO_TIP830_READ	Read input data of all channels

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.

Error code	Description
ENOTSUP	unknown ioctl() code

SEE ALSO

ioLib, basic I/O routine - ioctl()

4.3.1 FIO_TIP830_READ

This I/O control function starts a conversion on all channels of the module and reads the inout values. The function specific control parameter **arg** is a pointer on a *TIP830_READ_BUFFER*.

```
typedef struct
{
        unsigned int        flags;
        int                 data[TIP830_MAX_CHANNEL];
} TIP830_READ_BUFFER;
```

flags

There is only one flag that can be set for this function:

Flag	Description
TIP830_CORRECTION	If this flag is set the input data will be corrected with the correction data stored in the ID-Prom. If the flag is not set, the raw data will be returned.

data[]

This array will be filled with the ADC input values. The index specifies the channel number, index 0 specifies channel 1, index 1 specifies channel 2 and so on.
TIP830_MAX_CHANNEL (8) is defined in tip830.h and should not be changed.

The TIP830-10 is a 12 bit ADC module, the data value will be stored like 16-bit values (of the TIP830-20) in the range from -32768 to 32787.

The lower bits will be 0 for 12 bit values, except data correction is enabled than the lower bits may be set by data correction.

EXAMPLE

```
#include “tip830.h”

int                 fd;
TIP830_READ_BUFFER readBuf;
int                 retval;
int                 chan;

…

/*-------------------------
  Read corrected input data
  -------------------------*/
readBuf.flags = TIP830_CORRECTION;

…
```

…

```
retval = ioctl(fd, FIO_TIP830_READ, (int)&readBuf);
if (retval != ERROR)
{
     /* function succeeded */
     for (chan = 0; chan < TIP830_MAX_CHANNEL; chan++)
     {
          printf("Channel %d --- Value: %d\n",
                    chan + 1,
                    readBuf.data[chan]);
     }
}
else
{
     /* handle the error */
}
```

ERROR CODES

Error code	Description
ETIMEDOUT	The ADC conversion timed out