The Embedded I/O Company

TEWS TECHNOLOGIES

TIP570-SW-42

VxWorks Device Driver

16 Channel 12 Bit ADC and 8 Channel 12 Bit DAC

Version 3.0.x

User Manual

Issue 3.0.0

August 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

TIP570-SW-42

VxWorks Device Driver

16 Channel 12 Bit ADC and
8 Channel 12 Bit DAC

Supported Modules:
TIP570-10
TIP570-11

Issue	Description	Date
1.0	First Issue	July 19, 2000
1.1	General Revision	October 1, 2003
1.1.1	File list changed, new issue format	November 5, 2004
2.0.0	General Revision, New names for definitions, structures, and functions Carrier driver support, read() and write() are now handled in ioctl()	August 20, 2007
2.0.1	Carrier Driver description added	June 24, 2008
3.0.0	SMP Support, IPAC carrier interface functions removed	August 3, 2010

Table of Contents

1 Introduction

1.1 Device Driver

The TIP570-SW-42 VxWorks device driver software allows the operation of the supported IndustryPack 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 TIP570-SW-42 device driver supports the following features:

- reading ADC data with different input gains, with or without data correction
- support of single-ended and differential input lines
- set DAC output voltage with or without output data correction
- simultaneous DAC output for all channels
- Support for legacy and VxBus IPAC carrier driver
- SMP Support

The TIP570-SW-42 supports the modules listed below:

TIP570-10	16 Channel 12 Bit ADC (Gain: 1,2,5,10) and 8 Channel 12 Bit DAC	IndustryPack® compatible
TIP570-11	16 Channel 12 Bit ADC (Gain: 1,2,4,8) and 8 Channel 12 Bit DAC	IndustryPack® compatible

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

TIP570 User manual
TIP570 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 TIP570-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 'TIP570-SW-42':

tip570drv.c	TIP570 device driver source
tip570def.h	TIP570 driver include file
tip570.h	TIP570 include file for driver and application
tip570exa.c	Example application
include/ipac_carrier.h	Carrier driver interface definitions
TIP570-SW-42-3.0.0.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 TIP570-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: ./TIP570)

(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	3

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 tip570Drv()

NAME

tip570Drv() - installs the TIP570 driver in the I/O system

SYNOPSIS

#include “tip570.h”

STATUS tip570Drv(void)

DESCRIPTION

This function initializes and installs the TIP570 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  "tip570.h”

STATUS result;

/*-------------------
  Initialize Driver
  -------------------*/
result = tip570Drv();
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 tip570DevCreate()

NAME

tip570DevCreate() – Add a TIP570 device to the VxWorks system

SYNOPSIS

```
#include “tip570.h”

STATUS tip570DevCreate
(
        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 desired device instance beginning by 0. This parameter is 0 for the first TIP570 in the system, 1 for the second TIP570 and so forth. The order of TIP570 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  "tip570.h”

STATUS   result;

/*---------------------------------------------------------
  Create the device "/tip570/0" for the first TIP570 module
  ---------------------------------------------------------*/

result = tip570DevCreate("/tip570/0", 0, 0, NULL);

if (result == ERROR)
{
     /* 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 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
EISCONN	The device has already been created
ETIMEDOUT	An initial ADC Conversion has failed
ENXIO	No TIP570 module found for the specified parameter devIdx. If this error occurred for parameter devIdx=0, no TIP570 module at all was recognized.

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 TIP570 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 tip570DevCreate() must be used

flags

Not used

mode

Not used

EXAMPLE

```
int      fd;

/*------------------------------------------
  Open the device named "/tip570/0" for I/O
  ------------------------------------------*/

fd = open("/tip570/0", 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 “tip570.h”

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

DESCRIPTION

Special I/O operations 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_TIP570_READ	read ADC input value
FIO_TIP570_WRITE	set output of a single DAC channel
FIO_TIP570_SIM_WRITE	set output of all DAC 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	The specified function code is not supported

SEE ALSO

ioLib, basic I/O routine - ioctl()

4.3.1 FIO_TIP570_READ

This I/O control function starts an AD conversion with the specified parameters and will be blocked 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 ADC and DAC function can be used concurrently because of using different mutual-exclusion semaphores.

The function specific control parameter **arg** is a pointer to a TIP570_READ_BUFFER structure.

```
typedef struct
{
        unsigned long       flags;
        unsigned long       channel;
        unsigned long       gain;
        int                 data;
} TIP570_READ_BUFFER;
```

flags

This parameter is an ORed value of the following flags:

Flag	**Description**
TIP570_DIFF	If this flag is set the ADC input will use differential input interface. If this flag is not set, the ADC input will use single-ended interface.
TIP570_PIPL	If this flag is set the ADC will be used in pipeline mode.
TIP570_CORRECTION	If this flag is set the input value will be corrected with the manufacturer stored correction data.
TIP570_FAST	If this flag is set the driver will wait in a busy loop (polling-mode) until the conversion has finished instead of using interrupt notification. Depending on system performance in polling-mode the conversion time is about 9 microseconds instead of 15 microseconds in interrupt-mode. In interrupt-mode lower priority tasks were given a chance to run during conversion in polling-mode not.

channel

This value specifies the ADC input channel. Allowed channel numbers are 1..8 in differential mode, and 1..16 in single-ended mode.

gain

This value specifies the input gain. The allowed values depend on the version of the module. For TIP570-10 the allowed gain values are 1, 2, 5, and 10, for TIP570-11 allowed gain values are 1, 2, 4, and 8.

data

This is the parameter where the ADC input value will be stored. The value is returned as a 12-bit value in the range from -2048 to 2047.

EXAMPLE

```
#include “tip570.h”

int                fd;
TIP570_READ_BUFFER readBuf;
int                retval;

/*----------------------------------------------------------
  make conversion on ADC channel 3 (single-ended) with gain 2
  and return a corrected value
  ----------------------------------------------------------*/
readBuf.flags      = TIP570_CORRECTION | TIP570_FAST;
readBuf.channel    = 3;
readBuf.gain       = 2;

retval = ioctl(fd, FIO_TIP570_READ, (int)&readBuf);
if (retval != ERROR)
{
     /* function succeeded */
     printf(“ADC input value: %d\n”, readBuf.data);
}
else
{
     /* handle the error */
}
```

ERROR CODES

Error code	Description
EINVAL	An invalid parameter value has been specified. (gain, channel)
ENOTSUP	The detected model version is not supported
EBUSY	The module is already in use
ETIMEDOUT	Waiting for settling time, or AD conversion timed out

4.3.2 FIO_TIP570_WRITE

This I/O control function sets a new output value and starts DA conversion for a specified DAC channel. If a previous conversion is still in progress the driver will wait in a busy loop until the conversion has completed.

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

The ADC and DAC function can be used concurrently because of using different mutual-exclusion semaphores.

The function specific control parameter **arg** is a pointer to a TIP570_WRITE_BUFFER structure.

```
typedef struct
{
        unsigned long     flags;
        unsigned long     channel;
        int               data;
} TIP570_WRITE_BUFFER;
```

flags

This parameter is an ORed value of the following flags:

Flag	**Description**
TIP570_CORRECTION	If this flag is set, the output value will be corrected with the manufacturer stored correction data.

channel

This parameter selects the DAC channel. Valid channel numbers are 1 up to 8.

data

This value specifies the new output value. The value is a 12-bit value in the range from -2048 to 2047.

EXAMPLE

```
#include “tip570.h”

int                      fd;
TIP570_WRITE_BUFFER      writeBuf;
int                      retval;


/*------------------------------------------
  Set a new output voltage for DAC channel 5
  ------------------------------------------*/
writeBuf.flags     = 0;      /* no correction */
writeBuf.channel   = 5;
writeBuf.data      = 1024;   /* 5V */

retval = ioctl(fd, FIO_TIP570_WRITE, (int)&writeBuf);

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

ERROR CODES

Error code	Description
EINVAL	Invalid parameter value (channel)
ETIMEDOUT	The previous DA conversion has not been completed. A new DA conversion is blocked.
EBUSY	The module is already in use

4.3.3 FIO_TIP570_SIM_WRITE

This I/O control function sets new output values and starts DA conversion for all eight DAC channels. If a previous conversion is still in progress the driver will wait in a busy loop until the conversion has completed.

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

The ADC and DAC function can be used concurrently because of using different mutual-exclusion semaphores.

The function specific control parameter **arg** is a pointer to a TIP570_SIMWR_BUFFER structure.

```
typedef struct
{
    unsigned long      flags[TIP570_DAC_CHANNELS];
    int                data[TIP570_DAC_CHANNELS];
} TIP570_SIMWR_BUFFER;
```

flags[]

This parameter array specifies flags for DA conversion individually for each channel. The array index specifies the assigned channel, index 0 for channel 1, index 1 for channel 2, and so on. The flags are OR'ed values of the following flags:

Flag	Description
TIP570_CORRECTION	If this flag is set, the output value will be corrected with the manufacturer stored correction data.

data[]

This parameter array specifies data values for DA conversion individually for each channel. The array index specifies the assigned channel, index 0 for channel 1, index 1 for channel 2, and so on. The value is a 12-bit value in the range from -2048 to 2047.

EXAMPLE

```
#include “tip570.h”

int                     fd;
TIP570_SIMWR_BUFFER     simwrBuf;
int                     retval;
int                     channel;


/*--------------------------------------------------
  Set all DAC outputs to a value <channel_no> * 0x10
  even channels shall be corrected, odd channels not
  --------------------------------------------------*/
for (channel = 1; channel <= TIP570_DAC_CHANNELS; channel++)
{
     if (channel % 2)
     {
          simwrBuf.flags[channel - 1] = TIP570_CORRECTION;
     }
     else
     {
          simwrBuf.flags[channel - 1] = 0;
     }
     simwrBuf.data[channel - 1]  = channel * 0x10;
}

retval = ioctl(fd, FIO_TIP570_SIM_WRITE, (int)&simwrBuf);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

ERROR CODES

Error code	Description
ETIMEDOUT	A DA conversion has timed out.
EBUSY	The module is already in use.