The Embedded I/O Company

TEWS TECHNOLOGIES

TIP605-SW-42

VxWorks Device Driver

16 Digital Inputs

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

TIP605-SW-42

VxWorks Device Driver

16 Digital Inputs

Supported Modules:
TIP605

Issue	Description	Date
1.0	First Issue	October 1998
1.1	General Revision	October 2003
1.1.1	File list modified, supported BSPs	July 15, 2005
1.2.0	CARRIER-SW-42 and Intel x86 support	September 16, 2005
2.0.0	New tip605DevCreate() function, ChangeLog.txt added to file list, General Revision	September 26, 2008
2.1.0	SMP support	March 10, 2010

Table of Contents

1 Introduction

1.1 Device Driver

The TIP605-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(), read(),* and *ioctl()* functions.

The TIP605-SW-42 device driver supports the following features:

- reading the actual input value
- wait for selectable events (match, high-, low- or any-transition)
- program debounce time
- Support for legacy and VxBus IPAC carrier driver
- SMP Support

The TIP605-SW-42 supports the modules listed below:

TIP605-10	Digital Input for Avionic Applications	IndustryPack

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

TIP605 User manual
TIP605 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 TIP605-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.

2 Installation

Following files are located on the distribution media:

Directory path 'TIP605-SW-42':

tip605drv.c	TIP605 device driver source
tip605def.h	TIP605driver include file
tip605.h	TIP605include file for driver and application
tip605exa.c	Example application
include/ipac_carrier.h	Carrier driver interface definitions
TIP605-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 Driver configuration

The number of event wait jobs waiting for a specified event can be adapted to the applications requirements. There is a define of TIP605_MAX_JOBS in tip605.h, which specifies the maximum number of jobs that can be wait at same time for events. The basic value is 10.

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	---
Semaphores	maximum number of jobs	---

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 tip605Drv()

NAME

tip605Drv() - installs the TIP605 device driver in the I/O system and initialize the driver

SYNOPSIS

#include “tip605.h”

STATUS tip605Drv(void)

DESCRIPTION

This function installs the TIP605 driver in the I/O system, allocates driver resources and initializes them. The call of this function is the first thing we have to do, before adding any device to the system or performing any I/O request.

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 “tip605.h”

STATUS            result;

/*-------------------
  Initialize Driver
  -------------------*/
result = tip605Drv();
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 tip866DevCreate()

NAME

tip605DevCreate() – Add a TIP605 device to the VxWorks system

SYNOPSIS

```
#include “tip605.h”

STATUS tip605DevCreate
(
        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 (*TIP605_DEVCONFIG*) containing the default configuration of the device.

```
typedef struct
{
        struct ipac_resource    *ipac;
        long                    debounceTime;
} TIP605_DEVCONFIG;
```

ipac

Not used. Should be set to NULL.

debounceTime

Specifies the default input debounce time. The debounce time will be set in steps of 900µsec. (see TIP605 User manual)

EXAMPLE

```
#include “tip605.h”

STATUS             result;
TIP605_DEVCONFIG   devConfig;

/*-------------------------------------------------------
  Create the device "/tip605/0" for the first device
  Device specific parameters must be set up:
        debounce time = ~100ms (111)
  -------------------------------------------------------*/
devConfig.debounceTime =    111;

result = tip605DevCreate(   "/tip605/0",
                            0,
                            0,
                            (void*)&devConfig);
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, call tip605Drv() first
EINVAL	Unknown parameter value
EBUSY	The selected 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 TIP605 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 tip605DevCreate() must be used

flags

Not used

mode

Not used

EXAMPLE

```
int      fd;

/*----------------------------------------
  Open the device named "/tip605/0" for I/O
  ----------------------------------------*/
fd = open("/tip605/0", 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 read()

NAME

read() – read data from a specified device.

SYNOPSIS

```
int read
(
        int             fd,
        char            *buffer,
        size_t          maxbytes
)
```

DESCRIPTION

This routine reads the current state of the input lines.

PARAMETER

fd

This file descriptor specifies the device to be used. The file descriptor has been returned by the *open()* function.

buffer

This argument points to a user supplied buffer (unsigned short) where the function will store the value of the read input value.

maxbytes

This parameter specifies the maximum number of read bytes (buffer size). This value must be set to 2.

EXAMPLE

```
int                fd;
int                nbytes;
unsigned short     value;


/*--------------------------------
  Read input value of TIP605 device
  --------------------------------*/
retval = read(fd, (char*)&value, sizeof(value));
if (retval != ERROR)
{
     printf("%d bytes read\n", retval);
}
else
{
     /* handle the read error */
}
```

RETURNS

Number of bytes read 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) or a driver set error code described below.

Error code	Description
S_tip605_Drv_SM_BUFFER	The specified buffer size is to small

SEE ALSO

ioLib, basic I/O routine - read()

4.4 ioctl()

NAME

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

SYNOPSIS

#include “tip605.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
TIP605_EVREAD	Wait for event and read the current input lines
TIP605_DEBTIMESET	Set Debouncer time

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
S_tip605_Drv_ILLREQUEST	The request code is unknown

SEE ALSO

ioLib, basic I/O routine - ioctl()

4.4.1 TIP605_EVREAD

This I/O control function waits for a specified input event. The function specific control parameter **arg** is a pointer on a *TIP605_EVRD_PAR* structure.

```
typedef struct
{
        int             mode;
        unsigned short  mask;
        unsigned short  match;
        unsigned short  value;
        unsigned long   timeout;
} TIP605_EVRD_PAR;
```

mode

This parameter specifies the event the function should wait for. The following events are defined:

Event	Description
TIP605_MATCH	This mode waits until all bits, which are masked by *mask*, have the defined state of *match*. Note: It is not recommended to use this mode, because runtime may lead to miss of the event.
TIP605_HIGH_TR	This mode waits until the input line specified by *mask* changes from low to high. Note: Only one bit shall be selected in this mode.
TIP605_LOW_TR	This mode waits until the input line specified by *mask* changes from high to low. Note: Only one bit shall be selected in this mode.
TIP605_ANY_TR	This mode waits until the input line specified by *mask* changes its state. Note: Only one bit shall be selected in this mode.

mask

This parameter specifies the relevant bits to wait for.
For ‘match’-events the value masks the input and match value, before the values are compared. For ‘transition’-events only one bit should be set, that specifies the input line for the transition should take place.

match

This parameter specifies the value the function should wait for. Only used for ‘match’-events.

value

This field returns the value of the input lines, after the specified event has occurred. Note: The value is read when entering the interrupt service routine, the value may have changed since the event has occurred.

timeout

This value specifies the time the function is willing to wait for the event to occur. If the event doesn’t occur in the specified time the function returns with an appropriate error code. The timeout is specified in milliseconds.

EXAMPLE

```
#include “tip605.h”

int                fd;
int                retval;
TIP605_EVRD_PAR    argBuf;

/*-------------------------------------------------------------------
  wait until input port is ‘XXXX 1000 0001 XXXX’, timeout after 10s
  -------------------------------------------------------------------*/
argBuf.mode   = TIP605_MATCH;
argBuf.mask   = 0x0FF0;
argBuf.match  = 0x0810;
argBuf.timeout= 10000;
retval = ioctl(fd, TIP605_EVREAD, (int)&argBuf);
if (retval != ERROR)
{
     /* event occurred */
}
else
{
     /* handle the error */
}

…

/*-------------------------------------------------------------
  wait for a low to high transition on INPUT 3, timeout after 5s
  -------------------------------------------------------------*/
argBuf.mode   = TIP605_HIGH_TR;
argBuf.mask   = (1 << 3);
argBuf.timeout= 5000;
retval = ioctl(fd, TIP605_EVREAD, (int)&argBuf);
if (retval != ERROR)
{
     /* event occurred */
}
else
{
     /* handle the error */
}
```

ERROR CODES

Error code	Description
S_tip605_Drv_ILLEVRDMODE	Unknown event specified
S_tip605_Drv_NO_TR_SEL	No valid INPUT line specified
S_tip605_Drv_BAD_MEM	The job structure has been destroyed, this is a fatal error, the system shall be restarted
S_tip605_Drv_NO_FREE_JOB	There are no free jobs left (Think about increasing the maximum job definition *TIP605_MAX_JOBS*)

4.4.2 TIP605_DEBTIMESET

This I/O control function changes the debounce time. The function specific control parameter **arg** specifies the new value of the debounce register in steps of 900µs. A value of -1 disables the debouncer.

debounce time = (1 + value) * 900µs.

EXAMPLE

```
#include “tip605.h”

int                fd;
int                retval;

/*--------------------------------
  Select a debounce value of ~10ms
  --------------------------------*/
retval = ioctl(fd, TIP605_DEBTIMESET, 11);
if (retval != ERROR)
{
     /* event occurred */
}
else
{
     /* handle the error */
}
```