The Embedded I/O Company

TEWS TECHNOLOGIES

TIP150-SW-95

QNX-Neutrino Device Driver

1 or 2 channel synchro/resolver-to-digital converter

Version 2.0.x

User Manual

Issue 2.0.0

April 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

TIP150-SW-95

QNX-Neutrino Device Driver

1 or 2 channel
synchro/resolver-to-digital converter

Supported Modules:
TIP150

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.

Issue	Description	Date
1.0	First Issue	May 31, 2002
2.0.0	IPAC-Carrier support added, general revision	April 22, 2010

Table of Contents

1 Introduction

1.1 Device Driver

The TIP150-SW-95 QNX-Neutrino device driver allows the operation of a TIP150 synchro/resolver-to-digital converter IndustryPack® on QNX-Neutrino operating systems.

The TIP150 device driver is basically implemented as a user installable Resource Manager and started by the TEWS IPAC Carrier Driver (CARRIER-SW-95) if a TIP150 module was found during scanning of supported carrier boards.

The standard file (I/O) functions (open, close and devctl) provide the basic interface for opening and closing a file descriptor and for performing device I/O and control operations.

The TIP150 device driver supports the following features:

- reading current value and status of a specified channel
- reading current values and status of a both channels (only TIP150-4x)
- configure channel resolution

The TIP150-SW-95 device driver supports the modules listed below:

TIP150-3x	1 channel synchro/resolver-to-digital converter	(IndustryPack®)
TIP150-4x	2 channel synchro/resolver-to-digital converter	(IndustryPack®)

To obtain more information about the features and use of TIP150 devices, it is recommended to read the manuals listed below.

TIP150 User manual

TIP150 Engineering Manual

CARRIER-SW-95 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 other differences. Also, the varying byte ordering (big-endian versus little-endian) of CPU boards will cause problems when accessing the IndustryPack I/O and memory spaces.

To simplify the implementation of IPAC device drivers which should work with every 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-95 is part of this TIP150-SW-95 distribution. It is located in the directory CARRIER-SW-95 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-95 User Manual for a detailed description on how to install and setup the CARRIER-SW-95 device driver, and for a description of the TEWS TECHNOLOGIES IPAC Carrier Driver concept.

2 Installation

The following files are located on the distribution media:

Directory path 'TIP150-SW-95':

TIP150-SW-95-SRC.tar.gz	GZIP compressed archive with driver source code
TIP150-SW-95-2.0.0.pdf	PDF copy of this manual
ChangeLog.txt	Release history
Release.txt	Release information

The files have to be copied to the desired target directory for installation purposes.

The GZIP compressed archive TIP150-SW-95-SRC.tar.gz contains the following files and directories:

Directory path 'tip150':

driver/tip150.c	Device driver source
driver/tip150.h	Device driver and application include file
driver/tip150def.h	Device driver include file
driver/Makefile	Recursive multiplatform build tree
driver/common.mk	
driver/nto/Makefile	
driver/nto/x86/Makefile	
driver/nto/x86/dll/Makefile	
example/tip150exa.c	Example application
example/Makefile	Recursive multiplatform build tree
example/common.mk	
example/nto/Makefile	
example/nto/x86/Makefile	
example/nto/x86/o/Makefile	

For installation, copy the tar-archive TIP150-SW-95-SRC.tar.gz to /usr/src and extract all files (e.g tar -xzvf TIP150-SW-95-SRC.tar.gz).). Afterwards, the necessary directory structure for the automatic build and the source files are available underneath the new directory called tip150.

Change to the driver directory */usr/src/tip150/driver* and copy the header file *tip150.h* to */usr/include* allowing user application programs sharing the TIP150 driver interface definitions and data structures.

Before building a new device driver, the TEWS TECHNOLOGIES IPAC carrier driver must be installed properly, because this driver includes the header files *ipac_*.h*, which are part of the IPAC carrier driver distribution. Please refer to the IPAC carrier driver user manual in the directory path *CARRIER-SW-95* on the distribution media.

It is very important to extract the TIP150-SW-95-SRC.tar.gz in the /usr/src directory, because otherwise the automatic build with make will fail.

2.1 Build the device driver

Change to the */usr/src/tip150/driver* directory

Execute the Makefile

```
# make install
```

After successful completion the driver dynamic library *tip150.so* will be installed in the directory */lib/dll*.

2.2 Build the example application

Change to the */usr/src/tip150/example* directory

Execute the Makefile:

```
# make install
```

After successful completion, the example binary (*tip150exa*) will be installed in the */bin* directory.

2.3 Start the driver process

To start the TIP150 resource manager you have to start the TEWS TECHNOLOGIES IPAC carrier driver. The IPAC carrier driver detects installed TEWS IPAC modules automatically and loads the appropriate driver dynamic libraries.

```
# ipac_class &
```

The TIP150 resource manager registers a device for each TIP150 in the QNX-Neutrinos pathname space. The device file /dev/tip150_0 belongs to the first TIP150 found the device file /dev/tip150_1 to the second TIP150 and so forth (please refer to the IPAC carrier driver manual for detailed information of the module search order).

This pathname must be used in the application program to open a path to the desired TIP150 device.

For debugging purposes, you can start the IPAC carrier driver with the –V (verbose) option. Now the resource manager will print versatile information about TIP150 configuration and command execution on the terminal window. For further details about debugging, please see the IPAC carrier driver manual.

Make sure that only one instance of the ipac_class process is started.

3 I/O Functions

This chapter describes the interface to the device driver I/O system.

3.1 open()

NAME

open() - open a file descriptor

SYNOPSIS

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int open
(
        const char   *pathname,
        int          flags
)
```

DESCRIPTION

The *open()* function creates and returns a new file descriptor for a TIP150 device.

PARAMETER

pathname

Specifies the device to open.

flags

Controls how the file is to be opened. TIP150 devices must be opened *O_RDWR*.

EXAMPLE

```
int  fd;

fd = open(“/dev/tip150_0”, O_RDWR);
if (fd == -1)
{
     /* Handle error */
}
```

RETURNS

The normal return value is a non-negative integer file descriptor. In the case of an error, a value of –1 is returned. The global variable *errno* contains the detailed error code.

ERROR CODES

Returns only Neutrino specific error codes, see Neutrino Library Reference.

SEE ALSO

Library Reference - open()

3.2 close()

NAME

close() – close a file descriptor

SYNOPSIS

```
#include <unistd.h>

int close
(
        int             filedes
)
```

DESCRIPTION

The *close()* function closes a file.

PARAMETER

filedes

Specifies the file to close.

EXAMPLE

```
int  fd;

if (close(fd) != 0)
{
     /* handle close error conditions */
}
```

RETURNS

The normal return value is 0. In the case of an error, a value of –1 is returned. The global variable *errno* contains the detailed error code.

ERROR CODES

Returns only Neutrino specific error code, see Neutrino Library Reference.

SEE ALSO

Library Reference - close()

3.3 devctl()

NAME

devctl() – device control functions

SYNOPSIS

```
#include <sys/types.h>
#include <unistd.h>
#include <devctl.h>
#include <tip150.h>

int devctl
(
        int             filedes,
        int             dcmd,
        void            *data_ptr,
        size_t          n_bytes,
        int             *dev_info_ptr
)
```

DESCRIPTION

The *devctl()* function sends a control code directly to a device.

PARAMETER

filedes

Specifies the device to perform the requested operation.

dcmd

Specifies the control code for the operation. The following commands are defined (*tip150.h*):

Command	Description
DCMD_TIP150_CONFIG	Set channel resolution
DCMD_TIP150_READ_SNGL	Read actual value and state of a selected channel
DCMD_TIP150_READ_DBL	Read actual states and values of both channels (only TIP150-4x)

data_ptr

Depends on the command and will be described for each command in detail later in this chapter. Usually points to a buffer that passes data between the user task and the driver.

n_bytes

Depends on the command and will be described for each command in detail later in this chapter. Usually defines the size of the buffer pointed to by *data_ptr*.

dev_info_ptr

Is unused for the TIP150 driver and should be set to *NULL*.

RETURNS

On success, EOK is returned. In the case of an error, the appropriate error code is returned by the function (not in errno!).

ERRORS

Returns only Neutrino specific error codes, see Neutrino Library Reference.

Other function dependent error codes will be described for each *devctl()* code separately. Note, the TIP150 driver always returns standard QNX error codes.

SEE ALSO

Library Reference - devctl()

3.3.1 DCMD_TIP150_CONFIG

DESCRIPTION

This function sets the resolution of the specified channel. The new configuration is passed by a structure (*TIP150_CONFIG_BUF*) pointed to by the argument *data_ptr*. The argument n_bytes defines the size of the provided read buffer (sizeof(*TIP150_CONFIG_BUF*)).

```
typedef struct
{
        unsigned long       channel;
        unsigned char       config;
} TIP150_CONFIG_BUF, *PTIP150_CONFIG_BUF;
```

channel

Specifies the channel number. The allowed channel number is 1 for TIP150-3x and 1 or 2 for TIP150-4x

config

This argument specifies the new resolution. The following table shows the allowed values:

Value	**Description**
TIP150_RES_10BIT	The resolution is set to 10 bit
TIP150_RES_12BIT	The resolution is set to 12 bit
TIP150_RES_14BIT	The resolution is set to 14 bit
TIP150_RES_16BIT	The resolution is set to 16 bit

EXAMPLE

```
#include <tip150.h>

int                 fd;
int                 result;
TIP150_CONFIG_BUF   ConfBuf;


/* Set channel channel 1 resolution to 14 bit */
ConfBuf.channel     = 1;
ConfBuf.config      = TIP150_RES_14BIT;
result = devctl(fd,
                DCMD_TIP150_CONFIG,
                &ConfBuf,
                sizeof(ConfBuf),
                NULL);
if (result == EOK)
{
     /* Configuration successful */
}
else
{
     /* Error occurred */
}
```

ERRORS

EINVAL	Invalid argument. This error code is returned if either the size of the message buffer is too small, or the specified receive queue is out of range.

3.3.2 DCMD_TIP150_READ_SNGL

DESCRIPTION

This function reads the current value and state of a specified channel. The function specific argument *data_ptr* points to the data structure (*TIP150_READ_BUF*) and *n_bytes* specifies its length in bytes.

```
typedef struct
{
        unsigned long       channel;
        unsigned short      value;
        unsigned char       status;
} TIP150_READ_BUF, *PTIP150_READ_BUF;
```

channel

Specifies the channel number. The allowed channel number is 1 for TIP150-3x and 1 or 2 for TIP150-4x

value

This argument returns the current value of the specified channel.

status

This argument returns the current state of the channel. The following flags may be set:

Value	**Description**
TIP150_FL_BIT_ERR	The build in test failed
TIP150_FL_LOS_ERR	There is a “loss of the signal” detected

EXAMPLE

```
#include <tip150.h>

int                     fd;
int                     result;
TIP150_READ_BUF         ReadBuf;


/* Read channel 2 */
ReadBuf.channel    = 2;
result = devctl(fd,
                DCMD_TIP150_READ_SNGL,
                &ReadBuf,
                sizeof(TIP150_READ_BUF),
                NULL);
if (result == EOK)
{
     /* Read successful */
     printf(“value %d – status %Xh\n”, ReadBuf.value, ReadBuf.status);
}
else
{
     /* Handle error */
}
```

ERRORS

EINVAL	Invalid argument. This error code is returned if either the size of the message buffer is too small, or the specified receive queue is out of range.

3.3.3 DCMD_TIP150_READ_DBL

DESCRIPTION

This function reads the current value and state of both channels. The function specific argument *data_ptr* points to the data structure (*TIP150_DBL_READ_BUF*) and *n_bytes* specifies its length in bytes.

```
typedef struct
{
        unsigned short      value[2];
        unsigned char       status[2];
} TIP150_DBL_READ_BUF, *PTIP150_DBL_READ_BUF;
```

value[]

This array returns the current values of the channels. Index 0 specifies the value of channel 1 and index 1 specifies the value of channel 2.

status[]

This argument returns the current state of the channels. Index 0 specifies the state of channel 1 and index 1 specifies the state of channel 2. The following flags may be set:

Value	**Description**
TIP150_FL_BIT_ERR	The build in test failed
TIP150_FL_LOS_ERR	There is a “loss of the signal” detected

EXAMPLE

```
#include <tip150.h>

int                     fd;
int                     result;
TIP150_DBL_READ_BUF     DblRdBuf;


/* Read both channels */
result = devctl(fd,
                DCMD_TIP150_READ_DBL,
                &DblRdBuf,
                sizeof(DblRdBuf),
                NULL);
if (result == EOK)
{
    /* Read successful */
    printf(“Channel 1: value %d – status %Xh\n”,
           DblRdBuf.value[0],
           DblRdBuf.status[0]);
    printf(“Channel 2: value %d – status %Xh\n”,
           DblRdBuf.value[1],
           DblRdBuf.status[1]);
}
else
{
    /* Handle error */
}
```

ERRORS

EINVAL	Invalid argument. This error code is returned if either the size of the message buffer is too small, or the specified receive queue is out of range