
TIP150-SW-95

QNX-Neutrino Device Driver

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

User Manual

Issue 1.0 Version 1.0.0

May 2002

TEWS TECHNOLOGIES GmbH

Am Bahnhof 7
Phone: +49-(0)4101-4058-0
e-mail: info@tews.com

25469 Halstenbek / Germany
Fax: +49-(0)4101-4058-19
www.tews.com

TEWS TECHNOLOGIES LLC

1 E. Liberty Street, Sixth Floor
Phone: +1 (775) 686 6077
e-mail: usasales@tews.com

Reno, Nevada 89504 / USA
Fax: +1 (775) 686 6024
www.tews.com

TIP150-SW-95

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

QNX-Neutrino device driver

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.

©2002 by TEWS TECHNOLOGIES GmbH

| Issue | Description | Date |
|--------------|--------------------|--------------|
| 1.0 | First Issue | May 31, 2002 |

Table of Content

| | | |
|----------|--|----------|
| 1 | INTRODUCTION..... | 4 |
| 2 | INSTALLATION..... | 5 |
| | 2.1 Build the device driver | 5 |
| | 2.2 Build the example application | 5 |
| | 2.3 Build the carrier board initialization example | 6 |
| | 2.4 Start the driver process..... | 6 |
| 3 | DEVICE INPUT/OUTPUT FUNCTIONS | 8 |
| | 3.1 open() | 8 |
| | 3.2 close()..... | 9 |
| | 3.3 devctl() | 10 |
| | 3.3.1 DCMD_T150_CONFIG | 12 |
| | 3.3.2 DCMD_T150_READ_SINGL | 14 |
| | 3.3.3 DCMD_T150_READ_DBL | 16 |

1 Introduction

The TIP150-SW-95 QNX-Neutrino device driver allows the operation of a TIP150 1 or 2 Channel synchro/resolver-to-digital converter IP on QNX-Neutrino operating systems.

The TIP150 device driver is basically implemented as a user installable Resource Manager. 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.

Supported features:

- Read actual value and status of a specified channel
- Read actual values and status of both channels (only TIP150-4x)
- Configure channel resolution

This needs an initializing of the carrier board, (e.g. SBS-PCI40). This driver should also announce the physical base addresses of the IP-slots. An example using the SBS-PCI40 is attached to the driver. This initialization software must be run before the driver is started.

2 Installation

The software is delivered on a PC formatted 3½" HD diskette.

Following driver specific files are located on the diskette:

| | |
|---------------------|--|
| /driver/tip150.c | Driver source code |
| /driver/tip150.h | Driver interface definitions and data structures |
| /driver/tip150def.h | Device driver include |
| /driver/node.h | Queue management definitions |
| /driver/node.c | Queue management source code |
| /example/example.c | Example application |
| /pci40/* | SBS-PCI40 installation example |
| TIP150-SW-95.pdf | This manual in PDF format |

For installation create a new directory (e.g. *../tip150*) in the */usr/src* directory and copy the complete */driver* and */example* directories (with sub-directories and all files) from the distribution diskette into the new created project directory.

Note

It's absolute important to create the *tip150* project directory in the */usr/src* directory otherwise the automatic build with *make* will fail.

2.1 Build the device driver

1. Change to the */usr/src/tip150/driver* directory
2. Execute the Makefile

```
# make install
```

After successful completion the driver binary will be installed in the */bin* directory.

2.2 Build the example application

1. Change to the */usr/src/tip150/example* directory
2. Execute the Makefile

```
# make install
```

After successful completion the example binary (*t150exam*) will be installed in the */bin* directory.

2.3 Build the carrier board initialization example

1. Change to the `/usr/src/pci40` directory
2. Execute the Makefile

```
# make install
```

After successful completion the example binary (`pci40`) will be installed in the `/bin` directory.

2.4 Start the driver process

The carrier board initialization must be called before the driver is started. For example call the SBS-PCI40 initialization.

```
pci40
```

This initialization will printout the base addresses of I/O-, memory space and interrupt vector for each IP-slot.

To start the TIP150 device driver respective the TIP150 resource manager you have to enter the process name with optional parameter from the command shell or in the startup script.

```
tip150 -A<IOaddress> &
```

This will start the TIP150 resource manager with one module mounted at the specified `<IOaddress>`. (The address depends on the system, this address is printed out by the SBS-PCI40 initialization example).

For starting the TIP150 resource manager with more than one module, you have simply to add the additional IO-addresses behind the `-A` flag.

```
tip150 -A<IOaddress_0>,<IOaddress_1>,...,<IOaddress_n> &
```

The TIP150 Resource Manager registers created devices in the Neutrinos pathname space under following names.

```
/dev/tip150_0  
/dev/tip150_1  
...  
/dev/tip150_x
```

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

```
fd = open("/dev/tip150_0", O_RDWR);
```

For debugging you can start the TIP150 Resource Manager with the `-v` option. Now the Resource Manager will print versatile information about TIP150 configuration and command execution on the terminal window.

```
tip150 -v -A<IOaddress> &
```

3 Device Input/Output 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 the TIP150 named by *pathname*.

The flags argument 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);
```

RETURNS

The normal return value from open 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.

ERRORS

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 fildes)
```

DESCRIPTION

The **close** function closes the file descriptor *fildes*.

EXAMPLE

```
int    fd;

...

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

RETURNS

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

ERRORS

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>

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, specified by *filedes*, causing the corresponding device to perform the requested operation.

The argument *dcmd* specifies the control code for the operation.

The arguments *data_ptr* and *n_bytes* depends on the command and will be described for each command in detail later in this chapter. Usually *data_ptr* points to a buffer that passes data between the user task and the driver and *n_bytes* defines the size of this buffer.

The argument *dev_info_ptr* is unused for the TIP150 driver and should be set to NULL.

The following devctl command codes are defined in *TIP150.h* :

| Value | Meaning |
|-----------------------------|---|
| <i>DCMD_T150_CONFIG</i> | Set channel resolution |
| <i>DCMD_T150_READ_SINGL</i> | Read actual value and state of a selected channel |
| <i>DCMD_T150_READ_DBL</i> | Read actual states and values of both channels (only TIP150-4x) |

See behind for more detailed information on each control code.

Note

To use these TIP150 specific control codes the header file *TIP150.h* must be included in the application

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

ENOTTY Inappropriate I/O control operation. This error code is returned if the requested devctl function is unknown. Please check the argument *cmd*.

Other function dependant 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_T150_CONFIG

NAME

DCMD_T150_CONFIG - Read from ADC Input Channel

DESCRIPTION

This devctl function sets the resolution of the specified channel. A pointer to the callers buffer (*T150_CONFIG_BUF*) and the size of this structure is passed by the parameters *data_ptr* and *n_bytes* to the device.

The *T150_CONFIG_BUF* structure has the following layout:

```
typedef struct
{
    /* INPUT: */
    unsigned long   channel;      /* channel number: 1 or 2          */
    unsigned char   config;       /* configuration value             */

    /* OUTPUT: */
} T150_CONFIG_BUF, *PT150_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 |
|-------------------------|---------------------------------|
| <i>TIP150_RES_10BIT</i> | The resolution is set to 10 bit |
| <i>TIP150_RES_12BIT</i> | The resolution is set to 12 bit |
| <i>TIP150_RES_14BIT</i> | The resolution is set to 14 bit |
| <i>TIP150_RES_16BIT</i> | The resolution is set to 16 bit |

EXAMPLE

```
int          fd;
int          result;
T150_CONFIG_BUF  ConfBuf;

...

/* Set channel channel 1 resolution to 14 bit */
ConfBuf.channel = 1;
ConfBuf.config = TIP150_RES_14BIT;
result = devctl (fd,
                DCMD_T150_CONFIG,
                &ConfBuf,
                sizeof(ConfBuf),
                NULL);
if (result == EOK)
{
    /* Configuration successful */
}

...
```

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.

SEE ALSO

Library Reference - devctl()

3.3.2 DCMD_T150_READ_SNGL

NAME

DCMD_T150_READ_SNGL - Read from a specified channel

DESCRIPTION

This devctl function reads the actual value of the specified channel. A pointer to the callers buffer (*T150_READ_BUF*) and the size of this structure is passed by the parameters *data_ptr* and *n_bytes* to the device.

The *T150_READ_BUF* structure has the following layout:

```
typedef struct
{
    /* INPUT: */
    unsigned long    channel;          /* channel number: 1 or 2          */

    /* OUTPUT: */
    unsigned short   value;           /* actual value                    */
    unsigned char    status;         /* actual state                    */
} T150_READ_BUF, *PT150_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 actual value of the channel.

status

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

| value | description |
|--------------------------|--|
| <i>TIP150_FL_BIT_ERR</i> | The build in test failed |
| <i>TIP150_FL_LOS_ERR</i> | There is a loss of the signal detected |

EXAMPLE

```
int          fd;
int          result;
T150_READ_BUF  ReadBuf;

...

/* Read channel 2 */
ReadBuf.channel = 2;
result = devctl (fd,
                DCMD_T150_READ_SNGL,
                &ReadBuf,
                sizeof(ReadBuf),
                NULL);
if (result == EOK)
{
    /* Read successful */
    printf("value %d - status %Xh\n",
          ReadBuf.value,
          ReadBuf.status);
}

...
```

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.

SEE ALSO

Library Reference - devctl()

3.3.3 DCMD_T150_READ_DBL

NAME

DCMD_T150_READ_DBL - Read actual value from both channels

DESCRIPTION

This devctl function reads the actual values and states of both channels on the module. This function is only available for TIP150-4x modules. A pointer to the callers buffer (*T150_DBL_READ_BUF*) and the size of this structure is passed by the parameters *data_ptr* and *n_bytes* to the device.

The *T150_DBL_READ_BUF* structure has the following layout:

```
typedef struct
{
    /* INPUT: */

    /* OUTPUT: */
    unsigned short value[2]; /* actual values */
    unsigned char status[2]; /* actual states */
} T150_READ_BUF, *PT150_READ_BUF;
```

value[]

This array returns the actual values of the channels. The index 0 specifies channel 1 and index 1 specifies channel 2.

status[]

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

| value | description |
|--------------------------|--|
| <i>TIP150_FL_BIT_ERR</i> | The build in test failed |
| <i>TIP150_FL_LOS_ERR</i> | There is a loss of the signal detected |

EXAMPLE

```
int          fd;
int          result;
T150_DBL_READ_BUF  ReadBuf;

...

/* Read both channels */
result = devctl (fd,
                DCMD_T150_READ_DBL,
                &ReadBuf,
                sizeof(ReadBuf),
                NULL);
if (result == EOK)
{
    /* Read successful */
    printf("Channel 1: value %d - status %Xh\n",
          ReadBuf.value[0],
          ReadBuf.status[0]);
    printf("Channel 2: value %d - status %Xh\n",
          ReadBuf.value[1],
          ReadBuf.status[1]);
}

...
```

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.

SEE ALSO

Library Reference - devctl()