

---

# TIP600-SW-95

## QNX-Neutrino Device Driver

16 Chanel Digital Inputs

Version 1.0.x

## User Manual

Issue 1.0.0

October 2004

**TIP600-SW-95**

16 Channel Digital Inputs

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.

©2004 by TEWS TECHNOLOGIES GmbH

<b>Issue</b>	<b>Description</b>	<b>Date</b>
1.0.0	First Issue	October 18, 2004

## Table of Content

<b>1</b>	<b>INTRODUCTION.....</b>	<b>4</b>
<b>2</b>	<b>INSTALLATION.....</b>	<b>5</b>
	2.1 Build the device driver .....	5
	2.2 Build the example application .....	5
	2.3 Start the driver process.....	6
<b>3</b>	<b>DEVICE INPUT/OUTPUT FUNCTIONS .....</b>	<b>7</b>
	3.1 open() .....	7
	3.2 close().....	8
	3.3 devctl() .....	9
	3.3.1 DCMD_TIP600_READ.....	11
	3.3.2 DCMD_TIP600_SET_DEBOUNCER.....	12
	3.3.3 DCMD_TIP600_READ_MATCH_EV .....	14
	3.3.4 DCMD_TIP600_READ_POS_TRANS_EV .....	16
	3.3.5 DCMD_TIP600_READ_NEG_TRANS_EV .....	18
	3.3.6 DCMD_TIP600_READ_ANY_TRANS_EV .....	20

---

# **1 Introduction**

The TIP600-SW-95 QNX-Neutrino device driver allows the operation of a TIP600 16 Digital Inputs IP on QNX-Neutrino operating systems and requires the TEWS QNX-Neutrino IPAC Carrier Driver Software (CARRIER-SW-95).

The TIP600 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:

- Reading digital input value
- Setting up the input debouncer
- Wait for a pattern match or transition input event and read input value

## 2 Installation

The TIP600-SW-95 directory on the distribution media contains the following files:

TIP600-SW-95.pdf	This manual in PDF format
TIP600-SW-95.tar	Archive with driver source code

The archive TIP600-SW-95.tar contains the following files and directories:

Driver/tip600.c	Driver source code
Driver/tip600.h	Driver interface definitions and data structures (public)
Driver/tip600def.h	Device driver include file (private)
Driver/Makefile	Script for make utility to build and install the driver
Example/example.c	Example application
Example/Makefile	Script for make utility to build and install the example app.

In order to perform an installation, first login as *root* and copy the TAR archive to the */usr/src* directory and then extract all files and directories.

**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 is 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's absolute important to extract the TIP600-SW-95.tar in the */usr/src* directory otherwise the automatic build with make will fail.

### 2.1 Build the device driver

Change to the */usr/src/tip600/driver* directory

Execute the Makefile

```
# make install
```

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

### 2.2 Build the example application

Change to the */usr/src/tip600/example* directory

Execute the Makefile

```
# make install
```

After successful completion the example binary (*t600exam*) will be installed in the */bin* directory.

## 2.3 Start the driver process

To start the TIP000 Resource Manager (Device Driver) you only have to start the TEWS TECHNOLOGIES IPAC Carrier Driver. The Carrier Driver automatically detects installed TEWS IPs and dynamically loads the concerning modul(s).

The TIP600 Resource Manager registers a device for each TIP600 in the QNX-Neutrinos pathname space under following name, where n specifies the used IPAC slot number (please refer to the IPAC Carrier Driver Manual):

```
/dev/tip600_n
```

This pathname must be used in the application program to open a path to the desired TIP600 device.

For debugging you can start the IPAC Carrier Driver with the `-V` (verbose) option. Now the Resource Manager will print versatile information about TIP600 configuration and command execution on the terminal window. For further details about debugging see IPAC Carrier Driver Manual.

## **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 TIP600 named by *pathname*. The flags argument controls how the file is to be opened. TIP600 devices must be opened *O\_RDWR*.

#### **EXAMPLE**

```
int fd;

fd = open("/dev/tip600_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 filedes)
```

### DESCRIPTION

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

### 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 TIP600 driver and should be set to NULL.

The following devctl command codes are defined in *tip600.h* :

Value	Meaning
<i>DCMD_TIP600_READ</i>	Write digital output value
<i>DCMD_TIP600_SET_DEBOUNCER</i>	Setup input debouncer
<i>DCMD_TIP600_READ_MATCH_EV</i>	Wait for input pattern match event
<i>DCMD_TIP600_READ_POS_TRANS_EV</i>	Wait for positive input line transition event
<i>DCMD_TIP600_READ_NEG_TRANS_EV</i>	Wait for negative input line transition event
<i>DCMD_TIP600_READ_ANY_TRANS_EV</i>	Wait for any input line transition event

See behind for more detailed information on each control code.

**To use these TIP600 specific control codes the header file tip600.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 *dcmd*.

Other function dependant error codes will be described for each devctl code separately. Note, the TIP600 driver always returns standard QNX error codes.

## SEE ALSO

Library Reference - devctl()

### 3.3.1 DCMD\_TIP600\_READ

#### NAME

DCMD\_TIP600\_READ – Read digital input value

#### DESCRIPTION

This function reads from the input registers of the TIP600 associated with the file descriptor *filedes*. A pointer to an unsigned short is passed by *data\_ptr*. The argument size should always be `sizeof(unsigned short)` and is passed by *n\_bytes*.

Bit 0 of the read value corresponds to input line 1, Bit 1 corresponds to input line 2 and so on.

#### EXAMPLE

```

unsigned short    inval;
int               result;

...

/*
** Send request to the device driver
*/
result = devctl(fd, DCMD_TIP600_READ, &inval, sizeof(inval), NULL);
/*
** Check the result of the last device I/O operation
*/
if( result == EOK)
{
    printf("\nRead input successful\n");
    printf("    Input value: %04Xh\n", inval);
}
else
{
    printf("ioctl failed (Error = %d) : %s\n", result, strerror(result));
}

```

#### ERRORS

No errors.

#### SEE ALSO

Library Reference - `devctl()`

### 3.3.2 DCMD\_TIP600\_SET\_DEBOUNCER

#### NAME

DCMD\_TIP600\_SET\_DEBOUNCER – Setup input debouncer

#### DESCRIPTION

This function sets up the input debouncer of the TIP600 associated with the file descriptor *filedes*. A pointer to an unsigned long value is passed by *data\_ptr*. The argument size should always be `sizeof(unsigned long)` and is passed by *n\_bytes*.

The debouncer timer value should be specified in the unsigned long parameter. (For a description debouncer timer value, have a look to the TIP600 User Manual).

#### EXAMPLE

```
unsigned long    debval;
int             result;

...

debval = 100;    /* Setup debounce time to 100us */

/*
** Send request to the device driver
*/
result = devctl(fd, DCMD_TIP600_SET_DEBOUNCER,
               &debval, sizeof(debval), NULL);

/*
** Check the result of the last device I/O operation
*/
if( result == EOK)
{
    printf("\Setting up debouncer successful\n");
}
else
{
    printf("ioctl failed (Error = %d) : %s\n", result, strerror(result));
}
```

## **ERRORS**

*ECHRNG*

Specified debounce time is out of range

## **SEE ALSO**

Library Reference - devctl()

### 3.3.3 DCMD\_TIP600\_READ\_MATCH\_EV

#### NAME

DCMD\_TIP600\_READ\_MATCH\_EV – Wait for pattern match event and read input value

#### DESCRIPTION

This function reads from the input registers of the TIP600 associated with the file descriptor *filedes* after a specified pattern match event has occurred. A pointer to the data structure *TIP600\_READ\_EVENT\_BUF* is passed by *data\_ptr*. The argument size should always be `sizeof(TIP600_READ_EVENT_BUF)` and is passed by *n\_bytes*.

The data structure *TIP600\_READ\_EVENT\_BUF* is defined in *tip600.h*:

```
typedef struct tip600_read_event_buf
{
    unsigned short    mask;
    unsigned short    match;
    long              timeout;
    unsigned short    value;
} TIP600_READ_EVENT_BUF;
```

#### *mask*

This value specifies the mask which bits of the input value should be watched for the match event.

#### *match*

This value specifies the match value that should match to the input value.

#### *timeout*

This value specifies the function timeout in seconds.

#### *value*

This value returns the input value after the event has occurred.

## EXAMPLE

```
TIP600_READ_EVENT_BUF  evBuf;
int                    result;

...

evBuf.mask = 0x00FF;      /* Wait for pattern on input lines 1..8 */
evBuf.match = 0x0012;    /* Input lines 2,5 active, all other passive */
evBuf.timeout = 60;     /* Timeout after 60 seconds */

/*
** Send request to the device driver
*/
result = devctl(fd, DCMD_TIP600_READ_MATCH_EV,
               &evBuf, sizeof(evBuf), NULL);

/*
** Check the result of the last device I/O operation
*/
if( result == EOK)
{
    printf("\nMatch event occurred\n");
    printf("    Input value: %04Xh\n", evBuf.value);
}
else
{
    printf("ioctl failed (Error = %d) : %s\n", result, strerror(result));
}
```

## ERRORS

<i>ENOSPC</i>	No more free jobs available
<i>ETIMEDOUT</i>	Timeout has occurred

## SEE ALSO

Library Reference - devctl()

### 3.3.4 DCMD\_TIP600\_READ\_POS\_TRANS\_EV

#### NAME

DCMD\_TIP600\_READ\_POS\_TRANS\_EV – Wait for positive transition event and read input value

#### DESCRIPTION

This function reads from the input registers of the TIP600 associated with the file descriptor *filedes* after a low to high transition has occurred on a specified input line. A pointer to the data structure *TIP600\_READ\_EVENT\_BUF* is passed by *data\_ptr*. The argument size should always be `sizeof(TIP600_READ_EVENT_BUF)` and is passed by *n\_bytes*.

The data structure *TIP600\_READ\_EVENT\_BUF* is defined in *tip600.h*:

```
typedef struct tip600_read_event_buf
{
    unsigned short    mask;
    unsigned short    match;
    long              timeout;
    unsigned short    value;
} TIP600_READ_EVENT_BUF;
```

#### *mask*

This value masks the input bit the transition should occur on. Only one bit should be specified.

#### *match*

This value is unused for transition waits.

#### *timeout*

This value specifies the function timeout in seconds.

#### *value*

This value returns the input value after the event has occurred.

## EXAMPLE

```
TIP600_READ_EVENT_BUF  evBuf;
int                    result;

...

evBuf.mask = 0x0001;      /* Wait for a transition on input line 1 */
evBuf.timeout = 60;      /* Timeout after 60 seconds */

/*
** Send request to the device driver
*/
result = devctl(fd, DCMD_TIP600_READ_POS_TRANS_EV,
               &evBuf, sizeof(evBuf), NULL);

/*
** Check the result of the last device I/O operation
*/
if( result == EOK)
{
    printf("\nTransition occurred\n");
    printf("    Input value: %04Xh\n", evBuf.value);
}
else
{
    printf("ioctl failed (Error = %d) : %s\n", result, strerror(result));
}
}
```

## ERRORS

<i>ENOSPC</i>	No more free jobs available
<i>ETIMEDOUT</i>	Timeout has occurred

## SEE ALSO

Library Reference - devctl()

### 3.3.5 DCMD\_TIP600\_READ\_NEG\_TRANS\_EV

#### NAME

DCMD\_TIP600\_READ\_NEG\_TRANS\_EV – Wait for negative transition event and read input value

#### DESCRIPTION

This function reads from the input registers of the TIP600 associated with the file descriptor *filedes* after a high to low transition has occurred on a specified input line. A pointer to the data structure *TIP600\_READ\_EVENT\_BUF* is passed by *data\_ptr*. The argument size should always be `sizeof(TIP600_READ_EVENT_BUF)` and is passed by *n\_bytes*.

The data structure *TIP600\_READ\_EVENT\_BUF* is defined in *tip600.h*:

```
typedef struct tip600_read_event_buf
{
    unsigned short    mask;
    unsigned short    match;
    long              timeout;
    unsigned short    value;
} TIP600_READ_EVENT_BUF;
```

#### *mask*

This value masks the input bit the transition should occur on. Only one bit should be specified.

#### *match*

This value is unused for transition waits.

#### *timeout*

This value specifies the function timeout in seconds.

#### *value*

This value returns the input value after the event has occurred.

## EXAMPLE

```
TIP600_READ_EVENT_BUF  evBuf;
int                    result;

...

evBuf.mask = 0x0001;          /* Wait for a transition on input line 1 */
evBuf.timeout = 60;          /* Timeout after 60 seconds */

/*
** Send request to the device driver
*/
result = devctl(fd, DCMD_TIP600_READ_NEG_TRANS_EV,
                &evBuf, sizeof(evBuf), NULL);

/*
** Check the result of the last device I/O operation
*/
if( result == EOK)
{
    printf("\nTransition occurred\n");
    printf("    Input value: %04Xh\n", evBuf.value);
}
else
{
    printf("ioctl failed (Error = %d) : %s\n", result, strerror(result));
}
}
```

## ERRORS

<i>ENOSPC</i>	No more free jobs available
<i>ETIMEDOUT</i>	Timeout has occurred

## SEE ALSO

Library Reference - devctl()

### 3.3.6 DCMD\_TIP600\_READ\_ANY\_TRANS\_EV

#### NAME

DCMD\_TIP600\_READ\_ANY\_TRANS\_EV – Wait for transition event and read input value

#### DESCRIPTION

This function reads from the input registers of the TIP600 associated with the file descriptor *filedes* after a transition has occurred on a specified input line. A pointer to the data structure *TIP600\_READ\_EVENT\_BUF* is passed by *data\_ptr*. The argument size should always be `sizeof(TIP600_READ_EVENT_BUF)` and is passed by *n\_bytes*.

The data structure *TIP600\_READ\_EVENT\_BUF* is defined in *tip600.h*:

```
typedef struct tip600_read_event_buf
{
    unsigned short    mask;
    unsigned short    match;
    long              timeout;
    unsigned short    value;
} TIP600_READ_EVENT_BUF;
```

#### *mask*

This value masks the input bit the transition should occur on. Only one bit should be specified.

#### *match*

This value is unused for transition waits.

#### *timeout*

This value specifies the function timeout in seconds.

#### *value*

This value returns the input value after the event has occurred.

## EXAMPLE

```
TIP600_READ_EVENT_BUF  evBuf;
int                    result;

...

evBuf.mask = 0x0001;      /* Wait for a transition on input line 1 */
evBuf.timeout = 60;      /* Timeout after 60 seconds */

/*
** Send request to the device driver
*/
result = devctl(fd, DCMD_TIP600_READ_ANY_TRANS_EV,
               &evBuf, sizeof(evBuf), NULL);

/*
** Check the result of the last device I/O operation
*/
if( result == EOK)
{
    printf("\nTransition occurred\n");
    printf("    Input value: %04Xh\n", evBuf.value);
}
else
{
    printf("ioctl failed (Error = %d) : %s\n", result, strerror(result));
}
}
```

## ERRORS

<i>ENOSPC</i>	No more free jobs available
<i>ETIMEDOUT</i>	Timeout has occurred

## SEE ALSO

Library Reference - devctl()