

---

# TIP675-SW-92

## QNX4 Device Driver

48 TTL I/O Lines with Interrupts

Version 1.0.x

## User Manual

Issue 1.0.0

February 2005

**TIP675-SW-92**

48 TTL I/O Lines with Interrupts

QNX4 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.

©2005 by TEWS TECHNOLOGIES GmbH

<b>Issue</b>	<b>Description</b>	<b>Date</b>
1.0.0	First Issue	February 16, 2005

## 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 Start the driver process.....	5
<b>3</b>	<b>DRIVER INTERFACE .....</b>	<b>6</b>
	3.1 Find out the server process id.....	6
	3.2 Send I/O request.....	7
	3.2.1 TIP675_IN_BUFFER.....	7
	3.2.2 TIP675_OUT_BUFFER.....	8
<b>4</b>	<b>DRIVER FUNCTIONS.....</b>	<b>10</b>
	4.1 TIP675_READ .....	11
	4.2 TIP675_WRITE .....	13
	4.3 TIP675_SET_DIR .....	15
	4.4 TIP675_ENA_EXCLK.....	17
	4.5 TIP675_DIS_EXCLK .....	18
	4.6 TIP675_WAIT_TRANS.....	19
<b>5</b>	<b>APPENDIX.....</b>	<b>22</b>
	5.1 Status and Error codes.....	22

# **1 Introduction**

The TIP675-SW-92 QNX4 device driver software allows the operation of a TIP675 *48 TTL I/O Lines with Interrupts* IP on QNX4 operating systems with Intel and Intel-compatible x86 CPU, and TEWS TECHNOLOGIES or SBS PCI40A PCI based carrier boards.

The TIP675 device driver is basically a user-level server program. The interactions between client process and the TIP675 server process are realized via messages.

To start an I/O request the client process has to send an appropriate message, which contains a function code and optional parameter, to the server process. After the I/O operation has finished the server process replies the request message with a completion status and optional process data to caller.

The TIP675 device driver includes the following functions:

- Read input lines
- Write output lines
- Program direction of every I/O line
- Configure simultaneous update feature
- Wait for a transition at a single input line or a group of input lines (OR'ed)

## 2 Installation

The software is delivered on a PC formatted 3½" HD diskette, which contains the following files:

TIP675-SW-92-SRC.tar	TAR archive holding the driver source files
TIP675-SW-92.pdf	This manual in PDF format

Following files are located in the TAR archive:

tip675/tip675.c	Driver source code
tip675/tip675.h	Driver and application include file
tip675/tip675def.h	Driver include file
tip675/tip675isr.c	Driver interrupt source code
tip675/tpxxxhwdep.c	Hardware dependent functions
tip675/tpxxxhwdep.h	Include file for hardware dependent functions
tip675/makefile	Driver Makefile
tip675/example/*	Example application

For installation extract the TAR archive into a desired target directory.

### 2.1 Build the device driver

1. Change to the target directory, subdirectory *tip675/*
2. Execute the Makefile

```
# make
```

### 2.2 Start the driver process

After successful compilation, the driver can be started as follows:

```
./tip675 &
```

Now the driver process will start a search for the first TIP675 module and will use this. If you want to use a second TIP675, you have to specify an additional option during driver startup. The additional parameter `-I<index>` will specify the index of the module. If you want to start a process using the third available module, you have to call:

```
./tip675 -I2 &
```

The process will be started with a default priority of 19. If you want to use a different priority, you can specify this using the option `-P<priority>` while starting the driver process.

## 3 Driver Interface

Basically all communication between the application process (client) and the TIP675 device driver (server) is done via the QNX message system. An I/O request can be issued by sending a message with the *Send()* function to the appropriate server process. The server process performs all necessary processing for the requested command and finishes the request by sending a reply message to the client process.

To determine the necessary process id of the server process, the client program can invoke the *qnx\_name\_locate()* function. The server process has registered itself under a unique name (e.g. tip675/0).

Note that many system calls use data structures, which should be obtained in a program from appropriate header files. Necessary header files are listed with the system call synopsis.

The following section describes how you can determine the process id of the TIP675 server process and how you can send I/O requests to the driver.

### 3.1 Find out the server process id

Before an application program can send I/O request messages, the process id of the appropriate server process must be known. Because every server process registers itself under a unique name we can use this name to locate the server process.

A valid server name contains the module name (tip675) and a zero based index separated by a slash (/). The index is equal to the index specified as argument on command line when the server process was started. If no “/” argument was specified the index is always 0.

A valid server name for the first device looks like this:

```
tip675/0
```

The corresponding process id (pid) can be retrieved with the *qnx\_name\_locate()* function (see also C Library Reference).

#### EXAMPLE:

```
#include <sys/name.h>

...

pid_t    pid;

pid =    qnx_name_locate(    0, "tip675/0", 0, NULL);

if (pid == -1) {
    /* server process not found */
}

...
```

## 3.2 Send I/O request

In order to request the driver to perform I/O processing, you have to send a well-defined message (*pointed to by smsg*) with the kernel function *Send()* to the server process. After sending the message the application process becomes blocked (reply blocked) until the server process is able to complete the request.

After completion the application task receives a reply message (*pointed to by rmsg*) from the server and leaves the blocked state. The received message contains the return status of the I/O operation and optional process data (e.g. input port data).

For a detailed description of the kernel function *Send()* please refer to the *QNX C Library Reference Manual*.

The structures of the send message (*TIP675\_IN\_BUFFER*) and receive message (*TIP675\_OUT\_BUFFER*) are defined in *tip675.h* and have the following layout:

### 3.2.1 TIP675\_IN\_BUFFER

```
typedef struct
{
    int                command;
    union {
        T675_BUFFER   write;
        T675_BUFFER   direction;
        T675_EVRD_BUFFER event;
        unsigned long  clkedge;
    } u;
} TIP675_IN_BUFFER;
```

#### *command*

Specifies the control command of the operation. This value identifies the specific operation to be performed. Valid values are defined in *tip675.h* and described in detail in the next chapter.

#### *u.write*

This structure will be used for the *TIP675\_WRITE* command. The structure will be described in detail below together with the command description.

#### *u.direction*

This structure will be used for the *TIP675\_SET\_DIR* command. The structure will be described in detail below together with the command description.

#### *u.event*

This structure will be used for the *TIP675\_WAIT\_TRANS* command. The structure will be described in detail below together with the command description.

#### *u.clkedge*

This value be used for the *TIP675\_ENA\_EXCLK* command. The meaning of this value will be described in detail below together with the command description.

### 3.2.2 TIP675\_OUT\_BUFFER

```
typedef struct
{
    int                cmdStat;
    union {
        T675_BUFFER    read;
        T675_EVRD_BUFFER event;
    } u;
} TIP675_OUT_BUFFER;
```

#### *cmdStat*

Returns the error/status code of the I/O operation. If the function succeeds, the value of *ReturnStatus* is 0. If the function fails, the value of *cmdStat* is set to an error code (see also tip675.h).

#### *u.read*

Contains the data read from digital input lines. The structure will be described in detail below together with the TIP675\_READ command description.

#### *u.event*

Contains information about the occurred event. The structure will be described in detail below together with the TIP675\_WAIT\_TRANS command description.

## EXAMPLE

```
#include <sys/kernel.h>
#include "tip675.h"

pid_t          pid;
int            result;
TIP675_IN_BUFFER  smsg;    /* driver input parameter */
TIP675_OUT_BUFFER rmsg;    /* driver output parameter */

smsg.command = TIP675_READ;

/*
** Send request to the device driver
*/
result = Send(pid, &smsg, &rmsg, sizeof(smsg), sizeof(rmsg));
if( result == 0 )
{
    if( rmsg.cmdStat == EOK )
    {
        /* Send completed successfully */
    }
    else
    {
        /* write command failed */
    }
}
else
{
    printf( "Send failed --> Error = %d.\n", errno );
}
```

## 4 Driver Functions

To request the driver to perform the desired I/O operation the application process must send an appropriate message *TIP675\_IN\_BUFFER* to the device driver (server process). The *TIP675\_IN\_BUFFER* (smsg) contains a function code and optional parameter for the desired I/O operation.

The following functions are support by a TIP675 driver:

<b>Value</b>	<b>Meaning</b>
<i>TIP675_READ</i>	Read digital input values
<i>TIP675_WRITE</i>	Write digital output values
<i>TIP675_SET_DIR</i>	Configure I/O lines as input or output
<i>TIP675_ENA_EXCLK</i>	Enable external clock for simultaneous update
<i>TIP675_DIS_EXCLK</i>	Disable external clock for simultaneous update
<i>TIP675_WAIT_TRANS</i>	Wait for specific transition on input line

See behind for more detailed information on each control code.

## 4.1 TIP675\_READ

This function returns the line status from the TIP675 digital I/O lines.

The *command* code must be set in *TIP675\_IN\_BUFFER*. No additional input parameter will be used.

The *read* selection of union *u* in the reply buffer will be used for output values (see below).

typedef struct

```
{
    unsigned short    line_1_16;    /* values for input lines 1 to 16          */
    unsigned short    line_17_32;   /* values for input lines 17 to 32         */
    unsigned short    line_33_48;   /* values for input lines 33 to 48        */
} TIP675_BUFFER;
```

*line\_1\_16*

This parameter returns the status of input lines 1 to 16. Bit 0 corresponds to line 1, bit 1 corresponds to line 2 and so on.

*line\_17\_32*

This parameter returns the status of input lines 17 to 32. Bit 0 corresponds to line 17, bit 1 corresponds to line 18 and so on.

*line\_33\_48*

This parameter returns the status of input lines 33 to 48. Bit 0 corresponds to line 33, bit 1 corresponds to line 34 and so on.

## EXAMPLE

```

#include <sys/kernel.h>
#include "tip675.h"

pid_t          pid;
int            result;
TIP675_IN_BUFFER  smsg;    /* driver input parameter */
TIP675_OUT_BUFFER rmsg;    /* driver output parameter */

/*
** Read digital input values
*/
smsg.command = TIP675_READ;

/*
** Send request to the device driver
*/
result = Send(pid, &smsg, &rmsg, sizeof(smsg), sizeof(rmsg));
if( result == 0 )
{
    if( rmsg.cmdStat == EOK )
    {
        /* Send completed successfully */
        printf("lines 1-16 = 0x%04x\n", rmsg.u.read.line_1_16);
    }
    else
    {
        /* read command failed */
    }
}
else
{
    printf( "Send failed --> Error = %d.\n", errno );
}

```

## RETURNS

ReturnStatus	value	description
	0x00000000	OK

## 4.2 TIP675\_WRITE

This function sets the digital output lines to the specified values. The line status will change only if the corresponding lines are configured as output.

The *command* code must be set in *TIP675\_IN\_BUFFER*. The *read* selection of union *u* in the input buffer will be used for input values (see below).

The output buffer will not be used.

```
typedef struct
{
    unsigned short    line_1_16;    /* values for output lines 1 to 16          */
    unsigned short    line_17_32;   /* values for output lines 17 to 32         */
    unsigned short    line_33_48;   /* values for output lines 33 to 48        */
} TIP675_BUFFER;
```

*line\_1\_16*

This parameter holds the new value for output lines 1 to 16. Bit 0 corresponds to line 1, bit 1 corresponds to line 2 and so on.

*line\_17\_32*

This parameter holds the new value for output lines 17 to 32. Bit 0 corresponds to line 17, bit 1 corresponds to line 18 and so on.

*line\_33\_48*

This parameter holds the new value for output lines 33 to 48. Bit 0 corresponds to line 1, bit 33 corresponds to line 34 and so on.

## EXAMPLE

```
#include <sys/kernel.h>
#include "tip675.h"

pid_t          pid;
int            result;
TIP675_IN_BUFFER  smsg;    /* driver input parameter */
TIP675_OUT_BUFFER rmsg;    /* driver output parameter */

/*
** Write digital output values
*/
smsg.command = TIP675_WRITE;
smsg.write.line_1_16 = 0x1234;
smsg.write.line_17_32 = 0x5678;
smsg.write.line_33_48 = 0x9ABC;

/*
** Send request to the device driver
*/
result = Send(pid, &smsg, &rmsg, sizeof(smsg), sizeof(rmsg));
if( result == 0 )
{
    if( rmsg.cmdStat == EOK )
    {
        /* Send completed successfully */
    }
    else
    {
        /* read command failed */
    }
}
else
{
    printf( "Send failed --> Error = %d.\n", errno );
}
```

## RETURNS

ReturnStatus	value	description
	0x00000000	OK

## 4.3 TIP675\_SET\_DIR

With this function each of the 48 I/O lines may be individually set as input or output. To configure a line to be input, set the corresponding bit in the mask to 0.

The *command* code must be set in *TIP675\_IN\_BUFFER*. The *direction* selection of union *u* in the input buffer will be used for input values (see below).

The output buffer will not be used.

```
typedef struct
{
    unsigned short    line_1_16;    /* direction mask for I/O lines 1 to 16    */
    unsigned short    line_17_32;   /* direction mask for I/O lines 17 to 32   */
    unsigned short    line_33_48;   /* direction mask for I/O lines 33 to 48   */
} TIP675_BUFFER;
```

### *line\_1\_16*

This parameter holds the new direction for I/O lines 1 to 16. Bit 0 corresponds to line 1, bit 1 corresponds to line 2 and so on. 0 means input, 1 means output.

### *line\_17\_32*

This parameter holds the new direction for I/O lines 17 to 32. Bit 0 corresponds to line 17, bit 1 corresponds to line 18 and so on. 0 means input, 1 means output.

### *line\_33\_48*

This parameter holds the new direction for I/O lines 33 to 48. Bit 0 corresponds to line 33, bit 1 corresponds to line 34 and so on. 0 means input, 1 means output.

## EXAMPLE

```
#include <sys/kernel.h>
#include "tip675.h"

pid_t          pid;
int            result;
TIP675_IN_BUFFER  smsg;    /* driver input parameter */
TIP675_OUT_BUFFER rmsg;    /* driver output parameter */

/*
** Configure line 1-24 as input, line 25-48 as output
*/
smsg.command = TIP675_SET_DIR;
smsg.direction.line_1_16 = 0x0000;
smsg.direction.line_17_32 = 0xFF00;
smsg.direction.line_33_48 = 0xFFFF;

/*
** Send request to the device driver
*/
result = Send(pid, &smsg, &rmsg, sizeof(smsg), sizeof(rmsg));
if( result == 0 )
{
    if( rmsg.cmdStat == EOK )
    {
        /* Send completed successfully */
    }
    else
    {
        /* read command failed */
    }
}
else
{
    printf( "Send failed --> Error = %d.\n", errno );
}
```

## RETURNS

ReturnStatus	value	description
	0x00000000	OK

## 4.4 TIP675\_ENA\_EXCLK

This function enables the simultaneous update feature of the TIP675. Passing a 0 as a parameter to this function will cause the input and output values to be latched on the rising edge of the external clock. Passing a 1 to the driver will cause the latch operation to occur on the falling edge.

The *command* code must be set in *TIP675\_IN\_BUFFER*. The *exclk* selection of union *u* in the input buffer will be used for input values (see below), the output buffer will not be used.

### EXAMPLE

```
#include <sys/kernel.h>
#include "tip675.h"
pid_t          pid;
int            result;
TIP675_IN_BUFFER  smsg;    /* driver input parameter */
TIP675_OUT_BUFFER rmsg;    /* driver output parameter */

smsg.command = TIP675_ENA_EXCLK;
smsg.exclk = 0;
/*
** Send request to the device driver
*/
result = Send(pid, &smsg, &rmsg, sizeof(smsg), sizeof(rmsg));
if( result == 0 )
{
    if( rmsg.cmdStat == EOK )
    {
        /* Send completed successfully */
    }
    else
    {
        /* command failed */
    }
}
else
{
    printf( "Send failed --> Error = %d.\n", errno );
}
```

### RETURNS

ReturnStatus	value	description
	0x00000000	OK

## 4.5 TIP675\_DIS\_EXCLK

This function disables the simultaneous update feature of the TIP675.

The *command* code must be set in *TIP675\_IN\_BUFFER*, no additional parameter is used.

The output buffer will not be used.

### EXAMPLE

```
#include <sys/kernel.h>
#include "tip675.h"

pid_t          pid;
int            result;
TIP675_IN_BUFFER  smsg;    /* driver input parameter */
TIP675_OUT_BUFFER rmsg;    /* driver output parameter */

smsg.command = TIP675_DIS_EXCLK;
/*
** Send request to the device driver
*/
result = Send(pid, &smsg, &rmsg, sizeof(smsg), sizeof(rmsg));
if( result == 0 )
{
    if( rmsg.cmdStat == EOK )
    {
        /* Send completed successfully */
    }
    else
    {
        /* command failed */
    }
}
else
{
    printf( "Send failed --> Error = %d.\n", errno );
}
```

### RETURNS

ReturnStatus	value	description
	0x00000000	OK

## 4.6 TIP675\_WAIT\_TRANS

This function will be blocked until a specified transition on a selected input line has occurred or the maximum allowed time has elapsed. If more than one input line is selected, at least one transition must occur (logical OR) to finish this function. On success this function returns the contents of the input registers to the caller.

The *command* code must be set in *TIP675\_IN\_BUFFER*. The *event* selection of union *u* in the input buffer will be used for input values (see below).

The status of the input registers will be returned in *TIP675\_OUT\_BUFFER*. The *event* selection of union *u* in the reply buffer will be used for output values (see below).

```
typedef struct
{
    unsigned short    mask_1_16;    /* bit mask for I/O lines 1 to 16          */
    unsigned short    mask_17_32;   /* bit mask for I/O lines 17 to 32         */
    unsigned short    mask_33_48;   /* bit mask for I/O lines 33 to 48        */
    unsigned short    input_1_16;   /* input values for I/O lines 1 to 16 after event */
    unsigned short    input_17_32;  /* input values for I/O lines 17 to 32 after event */
    unsigned short    input_33_48;  /* input values for I/O lines 33 to 48 after event */
    unsigned short    status_1_16;   /* event source, I/O lines 1 to 16        */
    unsigned short    status_17_32; /* event source, I/O lines 17 to 32       */
    unsigned short    status_33_48; /* event source, I/O lines 33 to 48       */
    long              mode;          /* transition to occur                     */
    long              timeout;       /* timeout in seconds                      */
} T675_EVRD_BUFFER;
```

*mask\_1\_16, mask\_17\_32, mask\_33\_48*

These parameters contain a bit mask to select a certain bit position or a group of bits for an input transition detection. Bits  $2^0$  to  $2^{15}$  of *mask\_1\_16* correspond to Line 1 to 16, Bits  $2^0$  to  $2^{15}$  of *mask\_17\_32* correspond to Line 17 to 32. A certain input line can be selected by setting the corresponding bit position to 1.

*input\_1\_16, input\_17\_32, input\_33\_48*

These parameters receive the contents of the line input registers after the requested event has occurred. Please note that the input register isn't latched with the interrupt and depending on the interrupt latency the read to the input register is delayed.

*status\_1\_16, status\_17\_32, status\_33\_48*

These parameters receive a bit mask of the corresponding input lines, which can be used to determine the source of the event (interrupt). This is useful if more than one bit is selected. If the selected transition has occurred, the corresponding bit position contains a 1.

*mode*

Specifies the transition mode for this request

<i>T675_HIGH_TR</i>	In this mode the function will be blocked until a high transition occurred at the selected input line(s).
<i>T675_LOW_TR</i>	In this mode the function will be blocked until a low transition occurred at the selected input line(s).
<i>T675_ANY_TR</i>	In this mode the function will be blocked until a low or high transition occurred at the selected input line(s).

*timeout*

This parameter specifies the amount of time (in seconds) the caller is willing to wait for the occurrence of the requested transition. A value of -1 means wait indefinitely.

**EXAMPLE**

```
#include <sys/kernel.h>
#include "tip675.h"

pid_t          pid;
int            result;
TIP675_IN_BUFFER  smsg;    /* driver input parameter */
TIP675_OUT_BUFFER rmsg;    /* driver output parameter */

/*
** Wait for any transition on line 1 or 48
*/
smsg.command = TIP675_WAIT_TRANS;
smsg.event.mask_1_16    = 0x0001;
smsg.event.mask_17_32  = 0x0000;
smsg.event.mask_33_48  = 0x8000;

/*
** Send request to the device driver
*/
result = Send(pid, &smsg, &rmsg, sizeof(smsg), sizeof(rmsg));
if( result == 0 )
{
    if( rmsg.cmdStat == EOK )
    {
        /* Send completed successfully, the transition arrived */
        /* determine interrupt source */
        if (smsg.event.status_1_16 & 0x0001)
        {
            printf("Transition on line 1\n");
        }
    }
}
```

```
    }
    if (smsg.event.status_33_48 & 0x8000)
    {
        printf("Transition on line 48\n");
    }
}
else
{
    /* command failed */
    if ( rmsg.cmdStat == ETIMEDOUT )
    {
        printf("Timeout! No transition occurred.\n");
    }
}
}
else
{
    printf( "Send failed --> Error = %d.\n", errno );
}
```

## RETURNS

ReturnStatus	value	description
	0x00000000	OK

---

# 5 Appendix

## 5.1 Status and Error codes

The following values are returned in the reply buffer in the argument *cmdStat*. These values are only valid if the *Send()* function returns successfully. Error numbers are defined in the QNX header file *error.h*. The following additional status codes are defined in *tip675.h*.

Value	Description
TIP675_ERR_ICLKEDGE	An invalid external clockedge has been specified.
TIP675_ERR_ITRANS	The specified transition mode is invalid.
TIP675_ERR_NOREQ	No free request buffer available.