The Embedded I/O Company

TEWS TECHNOLOGIES

TIP670-SW-95

QNX-Neutrino Device Driver

Digital I/O

Version 1.0.x

User Manual

Issue 1.0.0

August 2008

TEWS TECHNOLOGIES GmbH
Am Bahnhof 7
25469 Halstenbek, Germany
www.tews.com
Phone: +49 (0) 4101 4058 0
Fax: +49 (0) 4101 4058 19
e-mail: info@tews.com

TEWS TECHNOLOGIES LLC
9190 Double Diamond Parkway,
Suite 127, Reno, NV 89521, USA
www.tews.com
Phone: +1 (775) 850 5830
Fax: +1 (775) 201 0347
e-mail: usasales@tews.com

TIP670-SW-95

QNX-Neutrino Device Driver

Digital I/O

Supported Modules:
TIP670

Issue	Description	Date
1.0.0	First Issue	August 27, 2008

Table of Contents

1 Introduction

1.1 Device Driver

The TIP670-SW-95 QNX-Neutrino device driver allows the operation of the TIP670 Digital I/O IndustryPack® on QNX-Neutrino systems and requires the TEWS TECHNOLOGIES QNX-Neutrino IPAC Carrier Driver Software (CARRIER-SW-95).

The TIP670 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.

The TIP670-SW-95 device driver supports the following features:

- reading the input register
- writing the output register
- programming polarity of inputs and outputs
- programming watchdog timeout
- waiting for a transition at a single input line or a group of input lines (OR'ed)

The TIP670-SW-95 device driver supports the modules listed below:

TIP670-10	8 isolated digital inputs, 8 isolated digital outputs	IndustryPack® compatible
TIP670-20	4 isolated digital inputs, 4 isolated digital outputs	IndustryPack® compatible

To get more information about the features and use of TIP670 devices it is recommended to read the manuals listed below.

TIP670 User manual

TIP670 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 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-82 is part of this TIP670-SW-95 distribution. It is located in 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-82 User Manual for a detailed description 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

Following files are located on the distribution media:

Directory path 'TIP670-SW-95':

TIP670-SW-95-SRC.tar.gz	GZIP compressed archive with driver source code
TIP670-SW-95-1.0.0.pdf	PDF copy of this manual
ChangeLog.txt	Release history
Release.txt	Release information

For installation the files have to be copied to the desired target directory.

The GZIP compressed archive TIP670-SW-95-SRC.tar.gz contains the following files and directories:

Directory path 'tip670':

driver/tip670.c	TIP670 device driver source
driver/tip670def.h	TIP670 driver include file
driver/tip670.h	TIP670 include file for driver and application
driver/Makefile	TIP670 driver makefile
api/tip670api.h	TIP670 API include file for application
api/tip670api.c	TIP670 API source file
example/exampleapp	Directory containing Interactive Example application (using driver functions)
example/readapp	Directory containing Console application for reading data (using API)
example/writeapp	Directory containing Console application for writing data (using API)
example/configapp	Directory containing Console application for configuring the device (using API)

Each example directory contains the following structure:

tip670*example*.c	Example application source file
common.mk	make parameters
Makefile	recursive makefile
nto/Makefile	recursive makefile
nto/x86/Makefile	recursive makefile
nto/x86/o/Makefile	recursive makefile

In order to perform an installation, copy TIP670-SW-95-SRC.tar.gz to */usr/src* and extract all files of the archive. (`tar –xzf TIP670-SW-95-SRC.tar.gz`)

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.

Its absolute important to extract the TIP670-SW-95-SRC.tar.gz in the /usr/src directory otherwise the automatic build with make will fail.

2.1 Build the device driver

Change to the */usr/src/tip670/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 desired example application sub-directory beneath */usr/src/tip670/example/*. Make sure that either symbolic links to the TIP670 Application Programming Interface source and header file (tip670api.c and tip670api.h) located in */usr/src/tip670/api/* are available in the example's directory, or simply copy the API files into the directory. The API must be compiled together with the provided example applications.

Execute the Makefile:

```
# make install
```

After successful completion the example binary (*tip670exa, tip670read, tip670write, tip670config*) will be installed in the */bin* directory.

2.3 Start the driver process

To start the TIP670 Resource Manager (Device Driver) you only have to start the TEWS TECHNOLOGIES IPAC Carrier Driver. The Carrier Driver automatically detects installed TEWS IPACs and dynamically loads the concerning module(s).

```
# ipac_class &
```

The TIP670 Resource Manager registers a device for each TIP670 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/tip670_n
```

This pathname must be used in the application program to open a path to the desired TIP670 device.

For debugging you can start the IPAC Carrier Driver with the –V (verbose) option. Now the Resource Manager will print versatile information about TIP670 configuration and command execution on the terminal window. For further details about debugging see IPAC Carrier Driver Manual.

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 TIP670 device.

PARAMETER

pathname

Specifies the device to open.

flags

Controls how the file is to be opened. TIP670 devices must be opened *O_RDWR*.

EXAMPLE

```
int  fd;

fd = open( “/dev/tip670_0”, O_RDWR );
if (fd == -1)
{
     /* Handle error */
}
```

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.

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 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.

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>

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 (*tip670.h*):

Command	Description
DCMD_TIP670_READ	Read current digital input value
DCMD_TIP670_EVENT_READ	Wait for an event, and return input port (blocked read)
DCMD_TIP670_WRITE	Write new digital output value
DCMD_TIP670_WRITE_MASK	Write new output value with bitmask
DCMD_TIP670_OUTPUTBITS_SET	Set single output lines
DCMD_TIP670_OUTPUTBITS_CLEAR	Clear single output lines
DCMD_TIP670_OUTPUTGET	Return current output value
DCMD_TIP670_READ_INP_POL	Read input port polarity configuration
DCMD_TIP670_READ_OUT_POL	Read output port polarity configuration
DCMD_TIP670_WRITE_INP_POL	Configure new input port polarity
DCMD_TIP670_WRITE_OUT_POL	Configure new output port polarity

DCMD_TIP670_ENABLE_WD	Setup and enable output port watchdog
DCMD_TIP670_DISABLE_WD	Disable output port watchdog
DCMD_TIP670_TRIGGER_WD	Trigger output port watchdog

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 by *data_ptr*.

dev_info_ptr

Is unused for the TIP670 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

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 TIP670 driver always returns standard QNX error codes.

SEE ALSO

Library Reference - devctl()

3.3.1 DCMD_TIP670_READ

DESCRIPTION

This function reads the current input value.

The function specific argument *data_ptr* points to an *unsigned char* data buffer and *n_bytes* specify its length in bytes. Bit 0 of the data value corresponds to the first input line, bit 1 corresponds to the second input line and so on. For TIP670-20, only the lower 4 bits are valid.

EXAMPLE

```
#include <tip670.h>

int                fd;
int                result;
unsigned char      inVal;

/* read input value */
result = devctl(   fd,
                   DCMD_TIP670_READ,
                   &inVal,
                   sizeof(inVal),
                   NULL
                   );
if (result == EOK)
{
     printf( “Value: %02X\n”, inVal );
}
else
{
     /* Handle error */
}
```

ERRORS

EINVAL	Invalid argument specified, or buffer too small.

3.3.2 DCMD_TIP670_EVENT_READ

DESCRIPTION

This function reads the contents of the input ports after a specified event occur. Possible events are rising or falling edge or both (any transition) at a specified input bit or a pattern match of masked input bits.

Transition detection and value reading is done by an interrupt service routine, and is affected by the system's interrupt latency, so this function should not be used for fast changing signals.

The function specific argument *data_ptr* points to a data structure (*TIP670_EVRD_BUFFER*) and *n_bytes* specify its length in bytes.

```
typedef struct
{
        unsigned char     value;
        unsigned char     mask;
        unsigned char     match;
        unsigned char     mode;
        long              timeout;
} TIP670_EVRD_BUFFER;
```

value

This parameter returns the contents of the input port when the event occurred. The value is read in the interrupt service function and may not represent the value at the moment the event has occurred. For TIP670-20, only the lower 4 bits are relevant.

mask

This parameter specifies a bit mask. A 1 value marks the corresponding bit position as relevant. Bit 0 of this value corresponds to the first input line, bit 1 corresponds to the second input line and so on. For TIP670-20, only the lower 4 bits are relevant.

match

This parameter specifies a pattern that must match to the contents of the input port. Only the bit positions specified by *mask* must compare to the input port. For TIP670-20, only the lower 4 bits are relevant.

mode

This parameter specifies the “event” mode for this read request

Value	Description
TIP670_MATCH	The driver reads the input port if the masked input bits match to the specified pattern. The input mask must be specified in the parameter *mask*. A 1 value in *mask* means that the input bit value “must-match” identically to the corresponding bit in the *match* parameter. Note that due to interrupt latency, this function may not detect very fast signal changes.
TIP670_HIGH_TR	The driver reads the input port if a high-transition at the specified bit position occurs. A 1 value in *mask* specifies the bit position of the input port. If you specify more than one bit position the events are OR’ed. That means the read is completed if a high-transition at least at one relevant bit position occur.
TIP670_LOW_TR	The driver reads the input port if a low-transition at the specified bit position occurs. A 1 value in *mask* specifies the bit position of the input port. If you specify more than one bit position the events are OR’ed. That means the read is completed if a low-transition at least at one relevant bit position occur.
TIP670_ANY_TR	The driver reads the input port if a transition (high or low) at the specified bit position occurs. A 1 value in *mask* specifies the bit position of the input port. If you specify more than one bit position the events are OR’ed. That means the read is completed if a transition at least at one relevant bit position occur.

timeout

Specifies the amount of time (in seconds) the caller is at least willing to wait for the specified event to occur. A value of -1 means wait indefinitely or no timeout.

EXAMPLE

```
#include <tip670.h>

int                     fd;
int                     result;
TIP670_EVRD_BUFFER      EventBuf;

/*
** Wait for a rising edge (high transition) at first input line
*/
EventBuf.mode = TIP670_HIGH_TR;
EventBuf.mask = 0x01;

result = devctl(   fd,
                   DCMD_TIP670_EVENT_READ,
                   &EventBuf,
                   sizeof(EventBuf),
                   NULL
                   );
if (result == EOK)
{
     printf( “Value after Event: %02X\n”, EventBuf.value );
}
else
{
     /* Handle error */
}
```

ERRORS

EINVAL	Invalid argument specified, or buffer too small.
ETIMEDOUT	The event has not occurred before the specified timeout.

3.3.3 DCMD_TIP670_WRITE

DESCRIPTION

This function sets the new output value.

The function specific argument *data_ptr* points to an *unsigned char* data buffer, and *n_bytes* specify its length in bytes. Bit 0 of the data value corresponds to the first output line, bit 1 corresponds to the second output line and so on. For TIP670-20, only the lower 4 bits are valid.

EXAMPLE

```
#include <tip670.h>

int                fd;
int                result;
unsigned char      outVal;

/*
** Write new output value
*/
outVal = 0x42;
result = devctl(   fd,
                   DCMD_TIP670_WRITE,
                   &outVal,
                   sizeof(unsigned char),
                   NULL
                   );
if (result == EOK)
{
     /* OK */
}
else
{
     /* Handle error */
}
```

ERRORS

EINVAL	Invalid argument specified, or buffer too small.

3.3.4 DCMD_TIP670_WRITE_MASK

DESCRIPTION

This function writes some data bits to the digital output.

The function specific argument *data_ptr* points to a data structure (*TIP670_WRITE_BUFFER*) and *n_bytes* specify its length in bytes.

```
typedef struct
{
        unsigned char       value;
        unsigned char       mask;
} TIP670_WRITE_BUFFER;
```

value

This value specifies the new output value, which will be written to the TIP670 output, in conjunction with a bitmask. Bit 0 of this value corresponds to the first output line, bit 1 corresponds to the second output line and so on. For TIP670-20, only the lower 4 bits are relevant.

mask

This parameter specifies the bitmask. Only active bits (1) will be written to the TIP670 output register. Bit 0 of this value corresponds to the first output line, bit 1 corresponds to the second output line and so on. For TIP670-20, only the lower 4 bits are relevant.

EXAMPLE

```
#include <tip670.h>

int                     fd;
int                     result;
TIP670_WRITE_BUFFER     WriteBuf;

/*
** only write the lowest bit
*/
WriteBuf.value = 0xf1;
WriteBuf.mask  = 0x01;

result = devctl(   fd,
                   DCMD_TIP670_WRITE_MASK,
                   &WriteBuf,
                   sizeof(WriteBuf),
                   NULL
                   );
if (result == EOK)
{
     /* OK */
}
else
{
     /* Handle error */
}
```

ERRORS

EINVAL	Invalid argument specified, or buffer too small.

3.3.5 DCMD_TIP670_OUTPUTBITS_SET

DESCRIPTION

This function sets single output lines leaving other output lines in the current state.

The function specific argument *data_ptr* points to an *unsigned char* buffer, and *n_bytes* specify its length in bytes. Bit 0 of this value corresponds to the first output line, bit 1 corresponds to the second output line and so on. For TIP670-20, only the lower 4 bits are relevant.

EXAMPLE

```
#include <tip670.h>

int                fd;
int                result;
unsigned char      outVal;

/*
** set output line 1 and 8
*/
outVal = 0x81;

result = devctl(   fd,
                   DCMD_TIP670_OUTPUTBITS_SET,
                   &outVal,
                   sizeof(outVal),
                   NULL
                   );
if (result == EOK)
{
     /* OK */
}
else
{
     /* Handle error ot timeout */
}
```

ERRORS

EINVAL	Invalid argument specified, or buffer too small.

3.3.6 DCMD_TIP670_OUTPUTBITS_CLEAR

DESCRIPTION

This function clears single output lines leaving other output lines in the current state.

The function specific argument *data_ptr* points to an *unsigned char* buffer, and *n_bytes* specify its length in bytes. Bit 0 of this value corresponds to the first output line, bit 1 corresponds to the second output line and so on. For TIP670-20, only the lower 4 bits are relevant.

EXAMPLE

```
#include <tip670.h>

int                fd;
int                result;
unsigned char      outVal;

/*
** clear output line 1 and 2
*/
outVal = 0x03;

result = devctl(   fd,
                   DCMD_TIP670_OUTPUTBITS_CLEAR,
                   &outVal,
                   sizeof(outVal),
                   NULL
                   );
if (result == EOK)
{
     /* OK */
}
else
{
     /* Handle error ot timeout */
}
```

ERRORS

EINVAL	Invalid argument specified, or buffer too small.

3.3.7 DCMD_TIP670_OUTPUTGET

DESCRIPTION

This function returns the current state of the output register. This value is retrieved from the device driver, no hardware access is performed. This function will not retrigger the output watchdog.

The function specific argument *data_ptr* points to an *unsigned char* data buffer and *n_bytes* specify its length in bytes. Bit 0 of the data value corresponds to the first output line, bit 1 corresponds to the second output line and so on. For TIP670-20, only the lower 4 bits are valid.

EXAMPLE

```
#include <tip670.h>

int                fd;
int                result;
unsigned char      outVal;

/*
** read output value
*/
result = devctl(   fd,
                   DCMD_TIP670_OUTPUTGET,
                   &outVal,
                   sizeof(outVal),
                   NULL
                   );
if (result == EOK)
{
     printf( “Value: %02X\n”, outVal );
}
else
{
     /* Handle error */
}
```

ERRORS

EINVAL	Invalid argument specified, or buffer too small.

3.3.8 DCMD_TIP670_READ_INP_POL

DESCRIPTION

This function reads the data path polarity setting of the input port.

The function specific argument *data_ptr* points to an *unsigned char* data buffer, and *n_bytes* specify its length in bytes. A 0 in a particular bit position specifies the corresponding bit path of the port as inverting. If a bit is read as 1, the data path is non-inverting. Bit 0 of the data value corresponds to the first input line, bit 1 corresponds to the second input line and so on. For TIP670-20, only the lower 4 bits are valid.

EXAMPLE

```
#include <tip670.h>

int                fd;
int                result;
unsigned char      inpPol;


/*
** Read input polarity
*/
result = devctl(   fd,
                   DCMD_TIP670_READ_INP_POL,
                   &inpPol,
                   sizeof(inpPol),
                   NULL
                   );
if (result == EOK)
{
     printf( “Polarity (bit mask, 0 = inverting) = %02Xh\n”, inpPol);
}
else
{
     /* Handle error */
}
```

ERRORS

EINVAL	Invalid argument specified, or buffer too small.

3.3.9 DCMD_TIP670_READ_OUT_POL

DESCRIPTION

This function reads the data path polarity setting of the output port.

The function specific argument *data_ptr* points to an *unsigned char* data buffer, and *n_bytes* specify its length in bytes. A 0 in a particular bit position specifies the corresponding bit path of the port as non-inverting. If a bit is read as 1, the data path is inverting. Bit 0 of the data value corresponds to the first output line, bit 1 corresponds to the second output line and so on. For TIP670-20, only the lower 4 bits are valid.

EXAMPLE

```
#include <tip670.h>

int                fd;
int                result;
unsigned char      outPol;

/*
** Read output polarity
*/
result = devctl(   fd,
                   DCMD_TIP670_READ_OUT_POL,
                   &outPol,
                   sizeof(outPol),
                   NULL
                   );
if (result == EOK)
{
     printf( “Polarity (bit mask, 1 = inverting) = %02Xh\n”, outPol);
}
else
{
     /* Handle error */
}
```

ERRORS

EINVAL	Invalid argument specified, or buffer too small.

3.3.10 DCMD_TIP670_WRITE_INP_POL

DESCRIPTION

This function configures the data path polarity of the input port.

The function specific argument *data_ptr* points to an *unsigned char* data buffer, and *n_bytes* specify its length in bytes. A 0 in a particular bit position specifies the corresponding bit path of the port as inverting. If a bit is read as 1, the data path is non-inverting. Bit 0 of the data value corresponds to the first input line, bit 1 corresponds to the second input line and so on. For TIP670-20, only the lower 4 bits are valid.

EXAMPLE

```
#include <tip670.h>

int                fd;
int                result;
unsigned char      inpPol;


/*
** Write input polarity: invert upper 4 bits (line 5 to line 8)
*/
outPol = 0x0f;
result = devctl(   fd,
                   DCMD_TIP670_WRITE_INP_POL,
                   &inpPol,
                   sizeof(inpPol),
                   NULL
                   );
if (result == EOK)
{
     /* OK */
}
else
{
     /* Handle error */
}
```

ERRORS

EINVAL	Invalid argument specified, or buffer too small.

3.3.11 DCMD_TIP670_WRITE_OUT_POL

DESCRIPTION

This function configures the data path polarity of the output port.

The function specific argument *data_ptr* points to an *unsigned char* data buffer, and *n_bytes* specify its length in bytes. A 0 in a particular bit position specifies the corresponding bit path of the port as non-inverting. If a bit is read as 1, the data path is inverting. Bit 0 of the data value corresponds to the first output line, bit 1 corresponds to the second output line and so on. For TIP670-20, only the lower 4 bits are valid.

EXAMPLE

```
#include <tip670.h>

int                fd;
int                result;
unsigned char      outPol;

/*
** Write output polarity: invert lower 4 bits (line 1 to line 4)
*/
outPol = 0x0f;
result = devctl(   fd,
                   DCMD_TIP670_WRITE_OUT_POL,
                   &outPol,
                   sizeof(outPol),
                   NULL
                   );
if (result == EOK)
{
     /* OK */
}
else
{
     /* Handle error */
}
```

ERRORS

EINVAL	Invalid argument specified, or buffer too small.

3.3.12 DCMD_TIP670_ENABLE_WD

DESCRIPTION

This function configures and enables the watchdog facility of the output port, which will disable the outputs, if there is no trigger access to the TIP670 within the configured time. The watchdog time is specified in steps of 0.5 microseconds, valid values are in the range from 1 to 65535. After driver startup, the watchdog is disabled.

The function specific argument *data_ptr* points to an *unsigned short* data buffer holding the watchdog time, and *n_bytes* specify its length in bytes.

EXAMPLE

```
#include <tip670.h>

int                fd;
int                result;
unsigned short     WatchdogTime;

/*
** Configure Output Watchdog to 30ms
*/
WatchdogTime = 60000;
result = devctl(   fd,
                   DCMD_TIP670_ENABLE_WD,
                   &WatchdogTime,
                   sizeof(WatchdogTime),
                   NULL
                   );
if (result == EOK)
{
     /* OK */
}
else
{
     /* Handle error */
}
```

ERRORS

EINVAL	Invalid argument specified, or buffer too small.

3.3.13 DCMD_TIP670_DISABLE_WD

DESCRIPTION

This function disables the watchdog facility of the output port.

The function does not use the parameters *data_ptr* and *n_bytes*, they must be set to 0.

EXAMPLE

```
#include <tip670.h>

int  fd;
int  result;

/*
** Disable Output Watchdog
*/
result = devctl(   fd,
                   DCMD_TIP670_DISABLE_WD,
                   NULL,
                   0,
                   NULL);
if (result == EOK)
{
     /* OK */
}
else
{
     /* Handle error */
}
```

ERRORS

EINVAL	Invalid argument specified, or buffer too small.

3.3.14 DCMD_TIP670_TRIGGER_WD

DESCRIPTION

This function triggers the watchdog facility of the output port. Reading the inputs or writing the outputs of the TIP670 will trigger the watchdog, too.

The function does not use the parameters *data_ptr* and *n_bytes*, they must be set to 0.

EXAMPLE

```
#include <tip670.h>

int  fd;
int  result;

/*
** Trigger Output Watchdog
*/
result = devctl(   fd,
                   DCMD_TIP670_TRIGGER_WD,
                   NULL,
                   0,
                   NULL
                   );
if (result == EOK)
{
     /* OK */
}
else
{
     /* Handle error */
}
```

ERRORS

EINVAL	Invalid argument specified, or buffer too small.

4 API Documentation

4.1 General Functions

4.1.1 tip670open()

Name

tip670open() – opens a device.

Synopsis

```
int tip670open
(
     char *DeviceName
);
```

Description

Before I/O can be performed to a device, a file descriptor must be opened by a call to this function.

Parameters

DeviceName

This parameter points to a null-terminated string that specifies the name of the device.

Example

```
#include “tip670api.h”
int FileDescriptor;

/*
** open file descriptor to device
*/
FileDescriptor = tip670open( “/dev/tip670_0” );
if (FileDescriptor < 0)
{
          /* handle open error */
}
```

RETURNS

A device descriptor number, or -1 if the function fails. An error code will be stored in *errno*.

ERROR CODES

The error codes are stored in *errno.*

The error code is a standard error code set by the I/O system.

4.1.2 tip670close()

Name

tip670close() – closes a device.

Synopsis

```
int tip670close
(
     int  FileDescriptor
);
```

Description

This function closes previously opened devices.

Parameters

FileDescriptor

This value specifies the file descriptor to the hardware module retrieved by a call to the corresponding open-function.

Example

```
#include “tip670api.h”
int FileDescriptor;
int result;

/*
** close file descriptor to device
*/
result = tip670close( FileDescriptor );
if (result < 0)
{
     /* handle close error */
}
```

RETURNS

EOK, or -1 if the function fails. An error code will be stored in *errno*.

ERROR CODES

The error codes are stored in *errno*.

The error code is a standard error code set by the I/O system.

4.2 Device Access Functions

4.2.1 tip670read()

Name

tip670read() – read current input value.

Synopsis

```
int tip670read
(
     int                FileDescriptor,
     unsigned char      *pInputValue
);
```

Description

This function reads the current input value of the specified device.

Parameters

FileDescriptor

This value specifies the file descriptor to the hardware module retrieved by a call to the corresponding open-function.

pInputValue

This value is a pointer to an unsigned char buffer which receives the current input value. Bit 0 of this value corresponds to the first input line, bit 1 corresponds to the second input line and so on. For TIP670-20, only the lower 4 bits are valid.

Example

```
#include “tip670api.h”
int           FileDescriptor;
int           result;
unsigned char InputValue;

/*
** read current input value
*/
result = tip670read( FileDescriptor, &InputValue );
if (result == EOK)
{
     printf( “Input Value: %02X\n”, InputValue );
} else {
     /* handle error */
}
```

RETURNS

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

ERROR CODES

All error codes a standard error codes set by the I/O system.

4.2.2 tip670eventRead()

Name

tip670eventRead() – wait for an event, and read input value

Synopsis

```
int tip670read
(
     int           FileDescriptor,
     int           EventMode,
     unsigned char Mask,
     unsigned char Match,
     long          Timeout,
     unsigned char *pInputValue
);
```

Description

This function waits for an event, and reads the input value of the specified device after this event has occurred. This function should not be used for fast changing devices, because event detection and reading the input value is highly affected by the system's interrupt latency.

Parameters

FileDescriptor

This value specifies the file descriptor to the hardware module retrieved by a call to the corresponding open-function.

EventMode

This parameter specifies the "event" mode for this read request

Value	Description
TIP670_MATCH	The driver reads the input port if the masked input bits match to the specified pattern. The input mask must be specified in the parameter *mask*. A 1 value in *mask* means that the input bit value "must-match" identically to the corresponding bit in the *match* parameter. Note that due to interrupt latency, this function may not detect very fast signal changes.

TIP670_HIGH_TR	The driver reads the input port if a high-transition at the specified bit position occurs. A 1 value in *mask* specifies the bit position of the input port. If you specify more than one bit position the events are OR'ed. That means the read is completed if a high-transition at least at one relevant bit position occur.
TIP670_LOW_TR	The driver reads the input port if a low-transition at the specified bit position occurs. A 1 value in *mask* specifies the bit position of the input port. If you specify more than one bit position the events are OR'ed. That means the read is completed if a low-transition at least at one relevant bit position occur.
TIP670_ANY_TR	The driver reads the input port if a transition (high or low) at the specified bit position occurs. A 1 value in *mask* specifies the bit position of the input port. If you specify more than one bit position the events are OR'ed. That means the read is completed if a transition at least at one relevant bit position occur.

Mask

This parameter specifies a bit mask. A 1 value marks the corresponding bit position as relevant. Bit 0 of this value corresponds to the first input line, bit 1 corresponds to the second input line and so on. For TIP670-20, only the lower 4 bits are relevant.

Match

This parameter specifies a pattern that must match to the contents of the input port. Only the bit positions specified by *mask* must compare to the input port. For TIP670-20, only the lower 4 bits are relevant.

Timeout

Specifies the amount of time (in seconds) the caller is at least willing to wait for the specified event to occur. A value of -1 means wait indefinitely or no timeout.

pInputValue

This value is a pointer to an unsigned char buffer which receives the input value. Bit 0 of this value corresponds to the first input line, bit 1 corresponds to the second input line and so on. For TIP670-20, only the lower 4 bits are valid.

Example

```
#include “tip670api.h”
int           FileDescriptor;
int           result;
unsigned char InputValue;

/*
** wait at least 5 seconds for any transition on first input line
*/
result = tip670eventRead(
              FileDescriptor,
              TIP670_ANY_TR,     /* EventMode                */
              (1 << 0),          /* Mask                     */
              0,                 /* Match                    */
              5,                 /* Timeout                  */
              &InputValue        /* pointer to input value   */
         );
if (result == EOK)
{
     printf( “Event occurred.\n” );
     printf( “Input Value after event: %02X\n”, InputValue );
} else {
     /* handle error */
}
```

RETURNS

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

ERROR CODES

ETIMEDOUT	The event has not occurred before the specified timeout.

All other error codes a standard error codes set by the I/O system.

4.2.3 tip670write()

Name

tip670write() – write new output value.

Synopsis

```
int tip670write
(
     int                FileDescriptor,
     unsigned char      OutputValue
);
```

Description

This function writes a new output value for the specified device. All output lines are affected, and will be set according to the specified output value.

Parameters

FileDescriptor

This value specifies the file descriptor to the hardware module retrieved by a call to the corresponding open-function.

OutputValue

This value specifies the new output value. Bit 0 of this value corresponds to the first output line, bit 1 corresponds to the second output line and so on. For TIP670-20, only the lower 4 bits are valid.

Example

```
#include “tip670api.h”
int           FileDescriptor;
int           result;

/*
** write new output value:
** set 2nd  (bit 1) and 7th (bit 6) output lines to ON,
** all other lines to OFF.
*/
result = tip670write(
              FileDescriptor,
              (1 << 1) | (1 << 6)
         );
if (result == EOK)
{
     /* OK */
} else {
     /* handle error */
}
```

RETURNS

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

ERROR CODES

All error codes a standard error codes set by the I/O system.

4.2.4 tip670writeMask()

Name

tip670writeMask() – write relevant bits of output value.

Synopsis

```
int tip670writeMask
(
     int                FileDescriptor,
     unsigned char      OutputValue,
     unsigned char      BitMask
);
```

Description

This function writes relevant bits of a new output value for the specified device.

Parameters

FileDescriptor

This value specifies the file descriptor to the hardware module retrieved by a call to the corresponding open-function.

OutputValue

This value specifies the new output value. Bit 0 of this value corresponds to the first output line, bit 1 corresponds to the second output line and so on. For TIP670-20, only the lower 4 bits are valid.

BitMask

This parameter specifies the bitmask. Only active bits (1) will be written to the TIP670 output register, all other output lines will be left unchanged. Bit 0 of this value corresponds to the first output line, bit 1 corresponds to the second output line and so on. For TIP670-20, only the lower 4 bits are relevant.

Example

```
#include “tip670api.h”
int           FileDescriptor;
int           result;

/*
** write new output value:
** set 2nd  (bit 1) output line to ON, and 7th (bit 6) output line to OFF.
** leave all other output lines unchanged.
*/
result = tip670writeMask(
              FileDescriptor,
              (1 << 1),
              (1 << 1) | (1 << 6)
         );
if (result == EOK)
{
     /* OK */
} else {
     /* handle error */
}
```

RETURNS

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

ERROR CODES

All error codes a standard error codes set by the I/O system.

4.2.5 tip670outputSet()

Name

tip670outputSet() – set single output lines to ON.

Synopsis

```
int tip670outputSet
(
     int                FileDescriptor,
     unsigned char      OutputValue
);
```

Description

This function sets single output lines to ON leaving other output lines in the current state.

Parameters

FileDescriptor

This value specifies the file descriptor to the hardware module retrieved by a call to the corresponding open-function.

OutputValue

This value specifies the new output value. Active (1) bits will set the corresponding output line to ON, unset (0) bits will not have an effect on the corresponding output lines. Bit 0 of this value corresponds to the first output line, bit 1 corresponds to the second output line and so on. For TIP670-20, only the lower 4 bits are valid.

Example

```
#include “tip670api.h”
int           FileDescriptor;
int           result;

/*
** write new output value:
** set 2nd  (bit 1) and 3rd (bit 2) output line to ON.
** leave all other output lines unchanged.
*/
result = tip670outputSet(
              FileDescriptor,
              (1 << 1) | (1 << 2)
         );
if (result == EOK)
{
     /* OK */
} else {
     /* handle error */
}
```

RETURNS

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

ERROR CODES

All error codes a standard error codes set by the I/O system.

4.2.6 tip670outputClear()

Name

tip670outputClear() – clear single output lines to OFF.

Synopsis

```
int tip670outputClear
(
     int                FileDescriptor,
     unsigned char      OutputValue
);
```

Description

This function clears single output lines to OFF leaving other output lines in the current state.

Parameters

FileDescriptor

This value specifies the file descriptor to the hardware module retrieved by a call to the corresponding open-function.

OutputValue

This value specifies the new output value. Active (1) bits will clear the corresponding output line to OFF, unset (0) bits will not have an effect on the corresponding output lines. Bit 0 of this value corresponds to the first output line, bit 1 corresponds to the second output line and so on. For TIP670-20, only the lower 4 bits are valid.

Example

```
#include “tip670api.h”
int           FileDescriptor;
int           result;

/*
** write new output value:
** clear 2nd  (bit 1) and 4th (bit 3) output line to OFF.
** leave all other output lines unchanged.
*/
result = tip670outputClear(
              FileDescriptor,
              (1 << 1) | (1 << 3)
         );
if (result == EOK)
{
     /* OK */
} else {
     /* handle error */
}
```

RETURNS

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

ERROR CODES

All error codes a standard error codes set by the I/O system.

4.2.7 tip670outputGet()

Name

tip670outputGet() – return last written output value.

Synopsis

```
int tip670outputGet
(
     int                FileDescriptor,
     unsigned char      *pOutputValue
);
```

Description

This function returns the last written output value. This value does not match the current output value, if the output lines have been disabled by the output watchdog. This function will not retrigger the output watchdog.

Parameters

FileDescriptor

This value specifies the file descriptor to the hardware module retrieved by a call to the corresponding open-function.

pOutputValue

This value is a pointer to an unsigned char buffer which receives the output value. Bit 0 of this value corresponds to the first output line, bit 1 corresponds to the second output line and so on. For TIP670-20, only the lower 4 bits are valid.

Example

```
#include “tip670api.h”
int           FileDescriptor;
int           result;
unsigned char OutputValue;

/*
** get output value
*/
result = tip670outputGet(
              FileDescriptor,
              &OutputValue
         );
if (result == EOK)
{
     printf( “Output Value: %02X\n”, OutputValue );
} else {
     /* handle error */
}
```

RETURNS

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

ERROR CODES

All error codes a standard error codes set by the I/O system.

4.2.8 tip670readInputPolarity()

Name

tip670readInputPolarity() – read input polarity.

Synopsis

```
int tip670readInputPolarity
(
     int                FileDescriptor,
     unsigned char      *pPolarity
);
```

Description

This function reads the data path polarity setting of the input port.

Parameters

FileDescriptor

This value specifies the file descriptor to the hardware module retrieved by a call to the corresponding open-function.

pPolarity

This value is a pointer to an unsigned char buffer which receives the input polarity value. Active (1) bits represent an inverted data path, unset (0) bits represent non-inverting data path. Bit 0 of the data value corresponds to the first input line, bit 1 corresponds to the second input line and so on. For TIP670-20, only the lower 4 bits are valid.

Example

```
#include “tip670api.h”
int           FileDescriptor;
int           result;
unsigned char Polarity;

/*
** read input polarity
*/
result = tip670readInputPolarity(
              FileDescriptor,
              &Polarity
         );
if (result == EOK)
{
     printf( “Polarity (bit mask, 1 = inverting) = %02Xh\n”, Polarity);
} else {
     /* handle error */
}
```

RETURNS

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

ERROR CODES

All error codes a standard error codes set by the I/O system.

4.2.9 tip670readOutputPolarity()

Name

tip670readOutputPolarity() – read output polarity.

Synopsis

```
int tip670readOutputPolarity
(
     int                FileDescriptor,
     unsigned char      *pPolarity
);
```

Description

This function reads the data path polarity setting of the output port.

Parameters

FileDescriptor

This value specifies the file descriptor to the hardware module retrieved by a call to the corresponding open-function.

pPolarity

This value is a pointer to an unsigned char buffer which receives the output polarity value. Active (1) bits represent an inverted data path, unset (0) bits represent non-inverting data path. Bit 0 of the data value corresponds to the first output line, bit 1 corresponds to the second output line and so on. For TIP670-20, only the lower 4 bits are valid.

Example

```
#include “tip670api.h”
int           FileDescriptor;
int           result;
unsigned char Polarity;

/*
** read output polarity
*/
result = tip670readoutputPolarity(
              FileDescriptor,
              &Polarity
         );
if (result == EOK)
{
     printf( “Polarity (bit mask, 1 = inverting) = %02Xh\n”, Polarity);
} else {
     /* handle error */
}
```

RETURNS

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

ERROR CODES

All error codes a standard error codes set by the I/O system.

4.2.10 tip670writeInputPolarity()

Name

tip670writeInputPolarity() – Configure new input port polarity.

Synopsis

```
int tip670writeInputPolarity
(
     int                FileDescriptor,
     unsigned char      Polarity
);
```

Description

This function configures the data path polarity setting of the input port.

Parameters

FileDescriptor

This value specifies the file descriptor to the hardware module retrieved by a call to the corresponding open-function.

pPolarity

This value specifies the new input polarity value. Active (1) bits represent an inverted data path, unset (0) bits represent non-inverting data path. Bit 0 of the data value corresponds to the first input line, bit 1 corresponds to the second input line and so on. For TIP670-20, only the lower 4 bits are valid.

Example

```
#include “tip670api.h”
int           FileDescriptor;
int           result;

/*
** configure input polarity:
** invert 7th (bit 6) and 8th (bit 7) input line
*/
result = tip670writeInputPolarity(
              FileDescriptor,
              (1 << 7) | (1 << 6)
         );
if (result == EOK)
{
     /* OK */
} else {
     /* handle error */
}
```

RETURNS

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

ERROR CODES

All error codes a standard error codes set by the I/O system.

4.2.11 tip670writeOutputPolarity()

Name

tip670writeOutputPolarity() – Configure new output port polarity.

Synopsis

```
int tip670writeOutputPolarity
(
     int                FileDescriptor,
     unsigned char      Polarity
);
```

Description

This function configures the data path polarity setting of the output port.

Parameters

FileDescriptor

This value specifies the file descriptor to the hardware module retrieved by a call to the corresponding open-function.

pPolarity

This value specifies the new output polarity value. Active (1) bits represent an inverted data path, unset (0) bits represent non-inverting data path. Bit 0 of the data value corresponds to the first output line, bit 1 corresponds to the second output line and so on. For TIP670-20, only the lower 4 bits are valid.

Example

```
#include “tip670api.h”
int           FileDescriptor;
int           result;

/*
** configure output polarity:
** invert 1st (bit 0) and 2nd (bit 1) output line
*/
result = tip670writeOutputPolarity(
              FileDescriptor,
              (1 << 1) | (1 << 0)
         );
if (result == EOK)
{
     /* OK */
} else {
     /* handle error */
}
```

RETURNS

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

ERROR CODES

All error codes a standard error codes set by the I/O system.

4.2.12 tip670configureWatchdog()

Name

tip670configureWatchdog() – Setup output port watchdog.

Synopsis

```
int tip670configureWatchdog
(
     int                FileDescriptor,
     unsigned short     WatchdogTime
);
```

Description

This function configures the data path polarity setting of the output port.

Parameters

FileDescriptor

This value specifies the file descriptor to the hardware module retrieved by a call to the corresponding open-function.

WatchdogTime

This value specifies the watchdog time in steps of 0.5 microseconds. To enable the output watchdog timer, valid values are from 1 to 65535. A value of 0 disables the watchdog feature.

Example

```
#include “tip670api.h”
int           FileDescriptor;
int           result;

/*
** enable watchdog and configure the watchdog time to 30ms
*/
result = tip670configureWatchdog(
              FileDescriptor,
              60000
         );
if (result == EOK)
{
     /* OK */
} else {
     /* handle error */
}
```

RETURNS

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

ERROR CODES

All error codes a standard error codes set by the I/O system.

4.2.13 tip670triggerWatchdog()

Name

tip670triggerWatchdog() – trigger output port watchdog.

Synopsis

```
int tip670triggerWatchdog
(
     int                FileDescriptor
);
```

Description

This function triggers the output watchdog. Reading the inputs or writing the outputs of the TIP670 will trigger the watchdog, too

Parameters

FileDescriptor

This value specifies the file descriptor to the hardware module retrieved by a call to the corresponding open-function.

Example

```
#include “tip670api.h”
int           FileDescriptor;
int           result;

/*
** trigger watchdog
*/
result = tip670triggerWatchdog(
              FileDescriptor
         );
if (result == EOK)
{
     /* OK */
} else {
     /* handle error */
}
```

RETURNS

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

ERROR CODES

All error codes a standard error codes set by the I/O system.