*The Embedded I/O Company*

TEWS TECHNOLOGIES

# TIP605-SW-95

## QNX-Neutrino Device Driver

16 Digital Inputs

Version 1.1.x

## User Manual

Issue 1.1.0

October 2009

**TEWS TECHNOLOGIES GmbH**

Am Bahnhof 7    25469 Halstenbek, Germany

Phone: +49 (0) 4101 4058 0    Fax: +49 (0) 4101 4058 19

e-mail: info@tews.com    www.tews.com

# TIP605-SW-95

QNX-Neutrino Device Driver

16 Digital Inputs

Supported Modules:
TIP605

| Issue | Description | Date |
|---|---|---|
| 1.0.0 | First Issue | July 21, 2009 |
| 1.1.0 | New file added to installation file list | October 23, 2009 |

# Table of Contents

# 1 Introduction

## 1.1 Device Driver

The TIP605-SW-95 QNX-Neutrino device driver allows the operation of a TIP605 Digital I/O IndustryPack® on QNX-Neutrino operating systems.

The TIP605 device driver is basically implemented as a user installable Resource Manager and started by the TEWS IPAC Carrier Driver (CARRIER-SW-95) if a TIP605 module was found during scanning of supported carrier boards.

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

The TIP605 device driver supports the following features:

- reading the actual port value
- waiting for selectable input events ( high-, low-, any-transition on the input port)
- setting programmable debounce times for digital filtering

The TIP605-SW-95 device driver supports the modules listed below:

| | | |
|---|---|---|
| TIP605-10 | 16 Digital Input (Optically Isolated) | (IndustryPack®) |
| TIP605-50 | 16 Digital Input (Optically Isolated) | (IndustryPack®) |
| TIP605-51 | 16 Digital Input (Optically Isolated) | (IndustryPack®) |
| TIP605-52 | 16 Digital Input (Optically Isolated) | (IndustryPack®) |

To obtain more information about the features and use of TIP605 devices, it is recommended to read the manuals listed below.

TIP605 User manual

TIP605 Engineering Manual

CARRIER-SW-95 User Manual

## 1.2 IPAC Carrier Driver

IndustryPack (IPAC) carrier boards have different implementations of the system to IndustryPack bus bridge logic, different implementations of interrupt and error handling and other differences. Also, the varying byte ordering (big-endian versus little-endian) of CPU boards will cause problems when accessing the IndustryPack I/O and memory spaces.

To simplify the implementation of IPAC device drivers which should work with every supported carrier board, TEWS TECHNOLOGIES has designed a so called Carrier Driver that hides all differences of different carrier boards under a well defined interface.

The TEWS TECHNOLOGIES IPAC Carrier Driver CARRIER-SW-95 is part of this TIP605-SW-95 distribution. It is located in the directory CARRIER-SW-95 on the corresponding distribution media.

This IPAC Device Driver requires a properly installed IPAC Carrier Driver. Due to the design of the Carrier Driver, it is sufficient to install the IPAC Carrier Driver once, even if multiple IPAC Device Drivers are used.

Please refer to the CARRIER-SW-95 User Manual for a detailed description on how to install and setup the CARRIER-SW-95 device driver, and for a description of the TEWS TECHNOLOGIES IPAC Carrier Driver concept.

# 2 Installation

The following files are located on the distribution media:

Directory path 'TIP605-SW-95':

| | |
|---|---|
| TIP605-SW-95-SRC.tar.gz | GZIP compressed archive with driver source code |
| TIP605-SW-95-1.1.0.pdf | PDF copy of this manual |
| ChangeLog.txt | Release history |
| Release.txt | Release information |

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

The GZIP compressed archive TIP605-SW-95-SRC.tar.gz contains the following files and directories:

Directory path 'tip605':

| | |
|---|---|
| driver/tip605.c | Device driver source |
| driver/tip605.h | Device driver and application include file |
| driver/tip605def.h | Device driver include file |
| driver/Makefile | Recursive multiplatform build tree |
| driver/common.mk | |
| driver/nto/Makefile | |
| driver/nto/x86/Makefile | |
| driver/nto/x86/dll/Makefile | |
| example/tip605exa.c | Example application |
| example/Makefile | Recursive multiplatform build tree |
| example/common.mk | |
| example/nto/Makefile | |
| example/nto/x86/Makefile | |
| example/nto/x86/o/Makefile | |

For installation, copy the tar-archive TIP605-SW-95-SRC.tar.gz to /usr/src and extract all files (e.g tar -xzvf TIP605-SW-95-SRC.tar.gz). ). Afterwards, the necessary directory structure for the automatic build and the source files are available underneath the new directory called tip605.

Change to the driver directory */usr/src/tip605/driver* and copy the header file *tip605.h* to */usr/include* allowing user application programs sharing the TIP605 driver interface definitions and data structures.

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

**It is very important to extract the TIP605-SW-95-SRC.tar.gz in the /usr/src directory, because otherwise the automatic build with make will fail.**

## 2.1 Build the device driver

Change to the */usr/src/tip605/driver* directory

Execute the Makefile

```
# make install
```

After successful completion the driver dynamic library *tip605.so* will be installed in the directory */lib/dll*.

## 2.2 Build the example application

Change to the */usr/src/tip605/example* directory

Execute the Makefile:

```
# make install
```

After successful completion, the example binary (*tip605exa*) will be installed in the */bin* directory.

## 2.3 Start the driver process

To start the TIP605 resource manager you have to start the TEWS TECHNOLOGIES IPAC carrier driver. The IPAC carrier driver detects installed TEWS IPAC modules automatically and loads the appropriate driver dynamic libraries.

```
# ipac_class &
```

The TIP605 resource manager registers a device for each TIP605 in the QNX-Neutrinos pathname space. The device file /dev/tip605_0 belongs to the first TIP605 found the device file /dev/tip605_1 to the second TIP605 and so forth (please refer to the IPAC carrier driver manual for detailed information of the module search order).

This pathname must be used in the application program to open a path to the desired TIP605 device.

For debugging purposes, you can start the IPAC carrier driver with the –V (verbose) option. Now the resource manager will print versatile information about TIP605 configuration and command execution on the terminal window. For further details about debugging, please see the IPAC carrier driver manual.

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

# 3 I/O Functions

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

## 3.1 open()

### NAME

open() - open a file descriptor

### SYNOPSIS

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

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

### DESCRIPTION

The *open()* function creates and returns a new file descriptor for a TIP605 device.

### PARAMETER

*pathname*

Specifies the device to open.

*flags*

Controls how the file is to be opened. TIP605 devices must be opened *O_RDWR*.

### EXAMPLE

```
int  fd;

fd = open( “/dev/tip605_0”, O_RDWR );
if (fd == -1)
{
     /* Handle error */
}
```

## RETURNS

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

## ERROR CODES

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

## SEE ALSO

Library Reference - open()

## 3.2 close()

### NAME

close() – close a file descriptor

### SYNOPSIS

#include <unistd.h>

```
int close
(
        int             filedes
)
```

### DESCRIPTION

The *close()* function closes a file.

### PARAMETER

*filedes*

Specifies the file to close.

### EXAMPLE

```
int  fd;

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

### RETURNS

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

## ERROR CODES

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

## SEE ALSO

Library Reference - close()

## 3.3 devctl()

### NAME

devctl() – device control functions

### SYNOPSIS

```
#include <sys/types.h>
#include <unistd.h>
#include <devctl.h>
#include <tip605.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 (*tip605.h*):

| Command | Description |
|---|---|
| DCMD_TIP605_READ | Read current state of the input port |
| DCMD_TIP605_EVENT_READ | Read the state of the input port after a specified event (state transition) has occurred. |
| DCMD_TIP605_SET_DEBOUNCE_TIME | Setup the debounce timer register |

*data_ptr*

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

*n_bytes*

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

*dev_info_ptr*

Is unused for the TIP605 driver and should be set to *NULL*.

## RETURNS

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

## ERRORS

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

Other function dependent error codes will be described for each *devctl()* code separately. Note, the TIP605 driver always returns standard QNX error codes.

## SEE ALSO

Library Reference - devctl()

## 3.3.1 DCMD_TIP605_READ

### DESCRIPTION

This function reads the current state of the input port into an unsigned short buffer pointed to by the argument *data_ptr*. The argument n_bytes defines the size of the provided read buffer (sizeof(unsigned short)).

### EXAMPLE

```
#include <tip605.h>

int                  fd;
int                  result;
unsigned short       value;


/* read input value */
result = devctl(   fd,
                   DCMD_TIP605_READ,
                   &value,
                   sizeof(value),
                   NULL);
if (result == EOK)
{
     printf("Input Port = %04Xh\n", value);
}
else
{
     /* Handle error */
}
```

### ERRORS

| | |
|---|---|
| EINVAL | Invalid argument specified, or buffer too small. |

## 3.3.2 DCMD_TIP605_EVENT_READ

### DESCRIPTION

Read the state of the input port after a specified event (state transition) has occurred. That means read will be blocked until the specified event occurs or the request times out. Events can be generated on the rising, falling edge or both (any transition) of a specified I/O line.

**There is a delay between the specified event and the input value read which is based on the system and OS dependent interrupt latency.**

The function specific argument *data_ptr* points to the data structure (*TIP605_EVRD_BUFFER*) and *n_bytes* specifies its length in bytes.

```
typedef struct
{
        unsigned short      value;
        unsigned short      mask;
        unsigned char       mode;
        long                timeout;
} TIP605_EVRD_BUFFER;
```

*value*

Returns the state of the input port after the event has occurred.

*mask*

The parameter “mask” specifies a bit mask to select input lines that shall be observed for the specified state transition. Bit 0 corresponds to input 1, bit 1 to input 2 and so forth.
To select a certain input line set the corresponding bit position to 1.

*mode*

This parameter specifies the state transition that must occur before read will be unblocked. If more than one bit position is set in mask the events are OR’ed. That means read will be completed if the state transition at least at one relevant bit position occurs.

| Value | Description |
|---|---|
| TIP605_HIGH_TR | The driver reads the input port if a high-transition at the specified bit position occurs. |
| TIP605_LOW_TR | The driver reads the input port registers if a low-transition at the specified bit position occurs |
| TIP605_ANY_TR | The driver reads the input port registers if a transition (high or low) at the specified bit position occurs |

*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 <tip605.h>

int                     fd;
int                     result;
TIP605_EVRD_BUFFER      eventBuf;

/*
** Wait for a rising edge (high transition) at input line 1
*/
eventBuf.mode      = TIP605_HIGH_TR;
eventBuf.mask      = 0x0001;
eventBuf.timeout   = 10;         /* 10 seconds */

result = devctl(   fd,
                   DCMD_TIP605_EVENT_READ,
                   &eventBuf,
                   sizeof(eventBuf),
                   NULL);
if (result == EOK)
{
     printf("Value = %04X\n", eventBuf.value);
}
else
{
     /* Handle error */
}
```

### ERRORS

| | |
|---|---|
| EINVAL | Invalid argument specified, or buffer too small. |
| ETIMEDOUT | The allowed time to finish the read request is elapsed. |

## 3.3.3 DCMD_TIP605_SET_DEBOUNCE_TIME

### DESCRIPTION

This function is used to program the debounce time in 255 steps of 1.024 ms in a range of 8 µs to 261ms. A value of 0 sets the debounce time to the minimum of 8 µs. This is also the default setup after driver start.

The function argument *data_ptr* points to an unsigned char buffer which passes the debounce value to the driver. The argument *n_bytes* specifies the size of this buffer (sizeof(unsigned char)).

### EXAMPLE

```
#include <tip605.h>

int           fd;
int           result;
unsigned char value;

/*
**   set maximum debounce time (261 ms)
*/
value = 255;

result = devctl(   fd,
                   DCMD_TIP605_SET_DEBOUNCE_TIME,
                   &value,
                   sizeof(value),
                   NULL);

if (result != EOK)
{
     printf("\nSet debounce time failed --> Error = %d\n", result);
}
```

### ERRORS

| | |
|---|---|
| EINVAL | Invalid argument specified, or buffer too small. |