The Embedded I/O Company

TEWS TECHNOLOGIES

TDRV003-SW-95

QNX6 - Neutrino Device Driver

16 (8) Digital Inputs

16 (8) Digital Outputs

Version 1.0.x

User Manual

Issue 1.0.0

November 2009

powerBridge Computer

Ehlbeek 15a
30938 Burgwedel
fon 05139-9980-0
fax 05139-9980-49

www.powerbridge.de
info@powerbridge.de

TEWS TECHNOLOGIES GmbH
Am Bahnhof 7 25469 Halstenbek, Germany
9 (0) 4101 4058 0 Fax: +49 (0) 4101 4058 19
il: info@tews.com www.tews.com

TDRV003-SW-95

QNX6-Neutrino Device Driver

16 (8) Digital Inputs
16 (8) Digital Outputs

Supported Modules:
TPMC670
TPMC671

Issue	Description	Date
1.0.0	First Issue	November 12, 2009

Table of Contents

1 Introduction

The TDRV003-SW-95 QNX-Neutrino device driver allows the operation of the TDRV003 digital I/O PMC devices on QNX-Neutrino operating systems.

The TDRV003 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 TDRV003-SW-95 device driver supports the following features:

- setting output lines
- reading state of input lines (immediate)
- reading state of input lines on a specified input event (transition, match)
- enable and disable output watchdog
- clearing state of output watchdog
- enable and disable input debouncer

The TDRV003-SW-95 device driver supports the modules listed below:

TPMC670-x0	16 digital inputs (24V) 16 digital outputs (24V, 0,5A)	(PMC)
TPMC670-x1	8 digital inputs (24V) 8 digital outputs (24V, 0,5A)	(PMC)
TPMC671-x0	16 digital inputs (24V) 16 digital high side switch outputs (24V, 0,5A)	(PMC)
TPMC671-x1	16 digital inputs (24V) 16 digital low side switch outputs (24V, 0,5A)	(PMC)

In this document all supported modules and devices will be called TDRV003. Specials for a certain device will be advised.

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

TPMC670 and/or TPMC671 User Manual
TPMC670 and/or TPMC671 Engineering Manual

2 Installation

Following files are located in the directory TDRV003-SW-95 on the distribution media:

TDRV003-SW-95-SRC.tar.gz	GZIP compressed archive with driver source code
TDRV003-SW-95-1.0.0.pdf	This manual in PDF format
ChangeLog.txt	Release history
Release.txt	Information about the Device Driver Release

The GZIP compressed archive TDRV003-SW-95-SRC.tar.gz contains the following files and directories:

Directory path 'tdrv003':

/driver/tdrv003.c	Driver source code
/driver/tdrv003.h	Definitions and data structures for driver and application
/driver/tdrv003def.h	Device driver include
/driver/Makefile	
/driver/common.mk	
driver/nto/*	Build path
/example/example.c	Example application
/example/Makefile	
/example/common.mk	
example/nto/*	Build path

For installation copy the tar-archive into the /usr/src directory and unpack it (e.g. `tar -xzvf TDRV003-SW-95-SRC.tar.gz`). After that the necessary directory structure for the automatic build and the source files are available underneath the new directory called *tdrv003*.

Change to the driver directory */usr/src/tdrv003/driver* and copy the header file *tdrv003.h* to */usr/include* allowing user application programs sharing the TDRV003 driver interface definitions and data structures.

It is absolutely important to extract the TDRV003 tar archive in the /usr/src directory. Otherwise the automatic build with make will fail.

2.1 Build the device driver

Change to the /usr/src/tdrv003/driver directory

Execute the Makefile:

```
# make install
```

After successful completion the driver binary (tdrv003) will be installed in the /bin directory.

2.2 Build the example application

Change to the /usr/src/tdrv003/example directory

Execute the Makefile:

```
# make install
```

After successful completion the example binary (***tdrv003exa***) will be installed in the /bin directory.

2.3 Start the driver process

To start the TDRV003 device driver, you have to enter the process name with optional parameter from the command shell or in the startup script.

```
tdrv003 [-v] &
```

The TDRV003 Resource Manager creates one device for every supported module and registers the created devices in the Neutrinos pathname space under following names.

```
/dev/tdrv003_0
/dev/tdrv003_1
…
/dev/tdrv003_x
```

The reference between the created device names and the physical devices depends on the search order of the PCI bus driver. The TDRV003 searches for supported devices in the following order: TPMC670, TPMC671.

The pathname must be used in the application program to open a path to the desired TDRV003 device.

```
fd = open("/dev/tdrv003_0", O_RDWR);
```

For debugging, you can start the TDRV003 Resource Manager with the –v option. Now the Resource Manager will print versatile information about TDRV003 configuration and command execution on the terminal window.

```
tdrv003 –v &
```

Make sure that only one instance of the device driver process is started.

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 TDRV003 device named by pathname. The flags argument controls how the file is to be opened. TDRV003 devices must be opened O_RDWR.

EXAMPLE

```
int fd;

fd = open("/dev/tdrv003_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.

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 TDRV003 driver and should be set to NULL.

The following devctl command codes are defined in *tdrv003.h*:

Symbol	Meaning
DCMD_TDRV003_READ	Read input state (on event)
DCMD_TDRV003_WRITE	Set output lines
DCMD_TDRV003_DEBENABLE	Setup and enable input debouncing
DCMD_TDRV003_DEBDISABLE	Disable input debouncing
DCMD_TDRV003_WDENABLE	Enable output watchdog
DCMD_TDRV003_WDDISABLE	Disable output watchdog
DCMD_TDRV003_WDRESET	Delete watchdog error

See behind for more detailed information on each control code.

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 TDRV003 driver always returns standard QNX error codes.

SEE ALSO

Library Reference - devctl()

3.3.1 DCMD_TDRV003_READ

NAME

DCMD_TDRV003_READ – Read input state, if specified after waiting for an event

DESCRIPTION

This function reads the state of the input lines. If it is configured the function will wait until a specified event occurs on selected input lines, before reading. A pointer to the callers read buffer (*TDRV003_READ_BUFFER*) must be passed by the parameters *data_ptr* and *n_bytes* to the device.

```
typedef struct
{
        int             mode;
        unsigned short  mask;
        unsigned short  match;
        int             timeout;
        unsigned short  value;
} TDRV003_READ_BUFFER, *PTDRV003_READ_BUFFER;
```

Remember interrupt latency:

All modes waiting for an event may return a value which is not identical to the value at the moment the event has occurred. There is a latency between the event occurrence and the call of the drivers ISR reading the input state.

Members

mode

This argument specifies the event to wait for. Symbols for the input mode are defined in tdrv003.h:

Define	Description
TDRV003_NOW	The driver reads the input port and returns immediately to the caller. The parameter *mask*, *match* and *timeout* are not relevant in this mode.
TDRV003_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 than the input bit value “must-match” identically to the corresponding bit in the *match* parameter. This mode is not recommended for quickly changing signals. The driver may miss very short matching states, because the input state may have changed again when the event handling is done in the ISR.

TDRV003_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. The parameter *match* is not relevant.
TDRV003_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. The parameter *match* is not relevant
TDRV003_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. The parameter *match* is not relevant

mask

Specifies the bit mask of relevant bits for the event. A 1 value marks the corresponding bit position as relevant. Bit 0 represents input line 1, bit 1 represents input line 2, and so on.

match

Specifies the pattern that must match to the state of the input port. Only the bit positions specified by *mask* are relevant for the event. Bit 0 represents input line 1, bit 1 represents input line 2, and so on.

timeout

Specifies the amount of time (in seconds) the caller is willing to wait for the specified event to occur.

value

This argument returns the state of the input (when the event has occurred). Bit 0 represents input line 1, bit 1 represents input line 2, and so on.

EXAMPLE

```
#include <tdrv003.h>

int                 fd;
int                 result;
TDRV003_READ_BUF    rdBuf;

/* --- read input state immediately --- */
rdBuf.mode          = TDRV003_NOW;

…
```

```
…

result = devctl(fd, DCMD_TDRV003_READ, &rdBuf, sizeof(rdBuf), NULL);

if (result == EOK)
{
     /* Read has been successful */
     printf("Input State: %04Xh\n", rdBuf.value);
}
else
{
     /* Read failed */
}

…

/* --- read input state on a low to high transition on input line 4 --- */
rdBuf.mode         = TDRV003_HIGH_TR;
rdBuf.mask         = 1 << 3;         /* input line 4 */
rdBuf.timeout      = 20;             /* 20 seconds */

result = devctl(fd, DCMD_TDRV003_READ, &rdBuf, sizeof(rdBuf), NULL);

if (result == EOK)
{
     /* Read has been successful */
     printf("Input State: %04Xh\n", rdBuf.value);
}
else
{
     /* Read failed */
}
```

ERRORS

EINVAL	Invalid argument. This error code is returned if the size of the message buffer is too small, or a specified is invalid.
ENOMEM	No memory available to allocated resources to handle the read command.
ETIMEDOUT	The allowed time to finish the read request has elapsed.

3.3.2 DCMD_TDRV003_WRITE

NAME

DCMD_TDRV003_WRITE - Write output value

DESCRIPTION

This function writes a new value for the output lines. A pointer to the callers write buffer (*TDRV003_WRITE_BUFFER*) must be passed by the parameters *data_ptr* and *n_bytes* to the device.

```
typedef struct
{
        unsigned short      mask;
        unsigned short      value;
} TDRV003_WRITE_BUFFER, *PTDRV003_WRITE_BUFFER;
```

Members

mask

This argument specifies the mask of bits which should be affected. A 1 value marks the corresponding bit position to be set, a 0 value masks the bit position as “do not change”. Bit 0 represents input line 1, bit 1 represents input line 2, and so on.

value

This argument specifies the new output value. Bit 0 represents input line 1, bit 1 represents input line 2, and so on.

EXAMPLE

```
#include <tdrv003.h>

int                     fd;
int                     result;
TDRV003_WRITE_BUFFER    wrBuf;

/* --- set output lines 1..8 --- */
wrBuf.mask          = 0x00FF;
wrBuf.value         = 0x1234;         /* only 0x34 will be written */

…
```

```
…

result = devctl(fd, DCMD_TDRV003_WRITE, &wrBuf, sizeof(wrBuf), NULL);

if (result == EOK)
{
     /* New output successfully written */
}
else
{
     /* Write failed */
}
```

ERRORS

EIO	The watchdog status is set and must be reset before a new value can be written

3.3.3 DCMD_TDRV003_DEBENABLE

NAME

DCMD_TDRV003_DEBENABLE – Setup and enable input debouncing

DESCRIPTION

This function sets up the debouncer time and enables the input debouncer. The function dependent parameters *data_ptr* and *n_bytes* specifies a buffer (unsigned short) which contains the new value of the debouncer register.

EXAMPLE

```
#include <tdrv003.h>

int                fd;
int                result;
unsigned short     debVal;

/* --- start debouncer with ~1ms debouncing --- */
debVal = 143;

result = devctl(fd, DCMD_TDRV003_DEBENABLE, &debVal, sizeof(debVal), NULL);

if (result == EOK)
{
     /* Debouuncer successfully started */
}
else
{
     /* Debouncer start failed */
}
```

3.3.4 DCMD_TDRV003_DEBDISABLE

NAME

DCMD_TDRV003_DEBDISABLE – Disable input debouncing

DESCRIPTION

This function disables the input debouncer. The function dependent parameters are not used.

EXAMPLE

```
#include <tdrv003.h>

int             fd;
int             result;

/* --- stop debouncinging --- */
result = devctl(fd, DCMD_TDRV003_DEBDISABLE, NULL, 0, NULL);

if (result != EOK)
{
     /* Debouuncer successfully stopped */
}
else
{
     /* Debouncer stopped failed */
}
```

3.3.5 DCMD_TDRV003_WDENABLE

NAME

DCMD_TDRV003_WDENABLE – Enable output watchdog

DESCRIPTION

This function enables the output watchdog. The function dependent parameters are not used.

The output must be rewritten (triggered) within 120ms, or the watchdog will set the output lines inactive and announce the watchdog error.

A watchdog error must be cleared with DCMD_TDRV003_WDRESET before a new write is allowed and output lines can be set again.

EXAMPLE

```
#include <tdrv003.h>

int             fd;
int             result;

/* --- enable output watchdog --- */
result = devctl(fd, DCMD_TDRV003_WDENABLE, NULL, 0, NULL);

if (result != EOK)
{
     /* Output watchdog successfully enabled*/
}
else
{
     /* Enable watchdog failed */
}
```

3.3.6 DCMD_TDRV003_WDDISABLE

NAME

DCMD_TDRV003_WDDISABLE – Disable output watchdog

DESCRIPTION

This function disables the output watchdog. The function dependent parameters are not used.

EXAMPLE

```
#include <tdrv003.h>

int             fd;
int             result;

/* --- disable output watchdog --- */
result = devctl(fd, DCMD_TDRV003_WDDISABLE, NULL, 0, NULL);

if (result != EOK)
{
     /* Output watchdog successfully disabled */
}
else
{
     /* Disable watchdog failed */
}
```

3.3.7 DCMD_TDRV003_WDRESET

NAME

DCMD_TDRV003_WDRESET – Resets the state of the output watchdog

DESCRIPTION

This function resets the state (watchdog error) of the output watchdog and allows new writes to the output. The function dependent parameters are not used.

EXAMPLE

```
#include <tdrv003.h>

int             fd;
int             result;

/* --- reset output watchdog error --- */
result = devctl(fd, DCMD_TDRV003_WDRESET, NULL, 0, NULL);

if (result != EOK)
{
     /* Status of watchdog successfully cleared */
}
else
{
     /* Clear of watchdog status failed */
}
```