The Embedded I/O Company

TDRV008-SW-95

QNX6-Neutrino Device Driver

3x 16 bit I/O Ports with 512 Word FIFO and Handshake

Version 1.0.x

User Manual

Issue 1.0.0

April 2007

TEWS TECHNOLOGIES GmbH		**TEWS TECHNOLOGIES LLC**	
Am Bahnhof 7	Phone: +49 (0) 4101 4058 0	9190 Double Diamond Parkway,	Phone: +1 (775) 850 5830
25469 Halstenbek, Germany	Fax: +49 (0) 4101 4058 19	Suite 127, Reno, NV 89521, USA	Fax: +1 (775) 201 0347
www.tews.com	e-mail: info@tews.com	www.tews.com	e-mail: usasales@tews.com

TDRV008-SW-95

QNX6-Neutrino Device Driver

3x 16 bit I/O Ports with 512 Word FIFO and Handshake

Supported Modules:
TPMC682

Issue	Description	Date
1.0.0	First Issue	April 4, 2007

Table of Contents

1 Introduction

The TDRV008-SW-95 QNX6-Neutrino device driver allows the operation of TDRV008 compatible devices on QNX6-Neutrino operating systems.

The TDRV008 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 TDRV008 device driver includes the following functions:

- buffered read and write of the 16-bit ports (0, 1 & 2) in pulsed or interlocked handshake mode
- configure ports (direction, mode and hardware timeout)
- hardware FIFO flush
- write to the 8-bit GPO port 5
- read from the 8-bit GPI port 4

The TDRV008-SW-95 device driver supports the modules listed below:

TPMC682	3 x 16 bit I/O Ports with 512 Word FIFO and Handshake	PMC

In this document all supported modules and devices will be called TDRV008. Specials for certain devices will be advised.

To get more information about the features and use of TPMC682 devices it is recommended to read the manuals listed below.

TPMC682 User manual

TPMC682 Engineering Manual

2 Installation

Following files are located on the distribution media:

Directory path '.\TDRV008-SW-95\':

TDRV008-SW-95-SRC.tar.gz	GZIP compressed archive with driver source code
TDRV008-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 TDRV008-SW-95-SRC.tar.gz contains the following files and directories:

Directory path './tdrv008/':

/driver/tdrv008.c	Driver source code
/driver/tdrv008.h	Definitions and data structures for driver and application
/driver/tdrv008def.h	Device driver include
/driver/node.c	Queue management source code
/driver/node.h	Queue management definitions
/driver/nto/*	Build path
/example/tdrv008exa.c	Example application
/example/nto/*	Build path

For installation copy the tar-archive into the /usr/src directory and unpack it (e.g. `tar –xvf TDRV008-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 tdrv008.

In order to perform an installation, extract all files of the archive TDRV008-SW-95-SRC.tar.gz to the /usr/src directory. Otherwise the automatic build with make will fail.

2.1 Build the device driver

Change to the /usr/src/tdrv008/driver directory

Execute the Makefile:

```
# make install
```

After successful completion the driver binary (tdrv008) will be installed in the /bin directory.

Build the example application

Change to the */usr/src/tdrv008/example* directory

Execute the Makefile:

```
# make install
```

After successful completion the example binary (***tdrv008exa***) will be installed in the /bin directory.

2.2 Start the driver process

To start the TDRV008 device driver respective you have to enter the process name with optional parameter from the command shell or in the startup script.

```
tdrv008 [-v] &
```

The TDRV008 Resource Manager registers created devices in the Neutrinos pathname space under following names.

```
/dev/tdrv008_0
/dev/tdrv008_1
…
/dev/tdrv008_x
```

This pathname must be used in the application program to open a path to the desired TDRV008 device.

```
fd = open("/dev/tdrv008_0", O_RDWR);
```

For debugging, you can start the TDRV008 Resource Manager with the –v option. Now the Resource Manager will print versatile information about TDRV008 configuration and command execution on the terminal window.

```
tdrv008 –v &
```

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 TDRV008 named by pathname. The flags argument controls how the file is to be opened. TDRV008 devices must be opened O_RDWR.

EXAMPLE

```
int fd;

fd = open("/dev/tdrv008_0", O_RDWR);
```

RETURNS

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

ERRORS

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

SEE ALSO

Library Reference - open()

3.2 close()

NAME

close() – close a file descriptor

SYNOPSIS

#include <unistd.h>

int close (int filedes)

DESCRIPTION

The close function closes the file descriptor *filedes*.

EXAMPLE

```
int fd;

...

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

RETURNS

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

ERRORS

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

SEE ALSO

Library Reference - close()

3.3 devctl()

NAME

devctl() – device control functions

SYNOPSIS

```
#include <sys/types.h>
#include <unistd.h>
#include <devctl.h>

int devctl
(
        int         filedes,
        int         dcmd,
        void        *data_ptr,
        size_t      n_bytes,
        int         *dev_info_ptr
)
```

DESCRIPTION

The devctl function sends a control code directly to a device, specified by *filedes*, causing the corresponding device to perform the requested operation.

The argument *dcmd* specifies the control code for the operation.

The arguments *data_ptr* and *n_bytes* depends on the command and will be described for each command in detail later in this chapter. Usually *data_ptr* points to a buffer that passes data between the user task and the driver and *n_bytes* defines the size of this buffer.

The argument *dev_info_ptr* is unused for the TDRV008 driver and should be set to NULL.

The following devctl command codes are defined in *tdrv008.h*:

Value	Description
DCMD_TDRV008_READ	Buffered read from a 16-bit handshake port
DCMD_TDRV008_WRITE	Buffered write to a 16-bit handshake port
DCMD_TDRV008_GETPORT	Get the state of the 8-bit GPI port #4
DCMD_TDRV008_SETPORT	Set the state of the 8-bit GPO port #5
DCMD_TDRV008_CONFPORT	Port setup
DCMD_TDRV008_FLUSHPORTS	Flush the hardware FIFOs of the 16-bit ports

See behind for more detailed information on each control code.

To use these TDRV008 specific control codes the header file tdrv008.h must be included in the application.

RETURNS

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

ERRORS

Returns only QNX-Neutrino specific error codes, see Neutrino Library Reference. For each devctl function the error codes are described separately.

SEE ALSO

Library Reference - devctl()

3.3.1 DCMD_TDRV008_READ

NAME

DCMD_TDRV008_READ – Buffered read from a 16-bit handshake port

DESCRIPTION

This TDRV008 control function reads 16-bit values from the FIFO of a given port. If data isn't available when calling this function a timeout is used to implement a blocking read. A pointer to the callers I/O buffer (*TDRV008_RW_BUFFER*) and the size of this structure are passed by the parameters *data_ptr* and *n_bytes* to the device.

The *TDRV008_RW_BUFFER* structure has the following layout:

```
typedef struct
{
        int                 portNo;
        unsigned long       flags;
        int                 timeout;
        int                 bufferSize;
        int                 validWords;
        unsigned short      data[TDRV008_FIFOSIZE];
} TDRV008_RW_BUFFER, *PTDRV008_RW_BUFFER;
```

Members

portNo

This parameter holds the port number to read from. Valid values are 0, 1 and 2.

flags

This parameter decides about non-blocking and blocking read requests. For non-blocking operation set *flags* to TDRV008_F_RW_NOWAIT. To initialize a blocking read request set *flags* to zero.

timeout

This parameter defines the read timeout with a given resolution of one second. So if you set it to 1 the driver should wait at least one second and worst case two seconds before terminating the certain read request.

bufferSize

This parameter defines the maximum count of words to read. The *data* storage has to be large enough to receive the specified amount of 16-bit words.

validWords

This in and out parameter holds the count of data words actually read. After read requests completion maybe not all data words given by *bufferSize* were received in the given time. For this purpose *validWords* is used for the read request result. If it is set to a value greater than 0 when starting the read request and blocking read is used then *validWords* is meant as an offset to the first element of the *data* buffer. So successive read requests that are partially completed can use the same I/O buffer until final read request completion.

data

This field parameter is used as data storage. It has a fixed size of TDRV008_FIFOSIZE words to match the hardware fifo size.

Example

```
#include “tdrv008.h”

int        fd;
int        result;
TDRV008_RW_BUFFER rwBuf;
int i;

rwBuf.portNo = 1;                // read from port 1
rwBuf.flags = 0;                 // blocking read
rwBuf.timeout = 5;               // wait at least 5 seconds
rwBuf.bufferSize = 177;          // we want to receive 177 words
rwBuf.validWords = 0;            // no data received yet
/* rwBuf.data[] will be filled with incoming data words */

result = devctl (  fd,                        // TDRV008 device handle
                   DCMD_TDRV008_READ,         // control code
                   &rwBuf,                    // I/O buffer
                   sizeof(rwBuf),
                   NULL);
if( result == EOK ) {
     // Process data, rwBuf.validWords is the real read result
     for (i = 0; i < rwBuf.validWords; i++)
     {
          printf(“@0x&04X: 0x%02X”, i, rwBuf.data[i]);
     }
     ...
}
else {
     // Process devctl() error
     ...
}
...
```

ERRORS

EFAULT	Invalid pointer to the user buffer.
EINVAL	An argument specified value is invalid.
ECHRNG	The port specified by I/O buffer member portNo doesn't exist.
EIO	The port was configured as output port. It has to be an input port to start a read request.
EBUSY	The certain port is in use.

SEE ALSO

Library Reference - devctl()

3.3.2 DCMD_TDRV008_WRITE

NAME

DCMD_TDRV008_WRITE – Buffered write to a 16-bit handshake port

DESCRIPTION

This TDRV008 control function writes 16-bit values to the specified buffered output port. A pointer to a callers I/O buffer (*TDRV008_RW_BUFFER*) and the size of this structure are passed by the parameters *data_ptr* and *n_bytes* to the device.

The *TDRV008_RW_BUFFER* structure has the following layout:

```
typedef struct
{
        int                 portNo;
        unsigned long       flags;
        unsigned long       timeout;
        int                 bufferSize;
        int                 validWords;
        unsigned short      data[TDRV008_FIFOSIZE];
} TDRV008_RW_BUFFER, *PTDRV008_RW_BUFFER;
```

Members

portNo

This parameter holds the port number of the handshake port to write on. Valid values are 0, 1 and 2.

flags

This parameter is not used for this IOCTL function.

timeout

This parameter defines the write timeout with a given resolution of one second. So if you set it to 1 the driver should wait at least one second and worst case two seconds before terminating the certain write request.

bufferSize

This parameter defines the count of words to write to the hardware FIFO of the certain handshake port.

validWords

This in and out parameter holds the count of data words actually written. In the case of a full FIFO the write process can't send more data words to the certain port and will block for *timeout* seconds. For this purpose *validWords* is used for the write request result. If it is set to a value greater than 0 when starting the write request then *validWords* is meant as an offset to the first element of the *data* buffer. Successive write requests that are partially completed can use the same I/O buffer until final write request completion.

data

This field parameter is used as data source during the handshake transmission. It has a fixed maximum size of TDRV008_FIFOSIZE words to match the hardware FIFO size.

Example

```
#include “tdrv008.h”

...

int       fd;
int       result;
TDRV008_RW_BUFFER rwBuf;
int i;

rwBuf.portNo = 0;                // read from port 1
rwBuf.timeout = 2;               // wait at least 2 seconds
rwBuf.bufferSize = 411;          // we want to send 411 words
rwBuf.validWords = 0;            // start with element 0

for (i = 0; i < rwBuf.bufferSize; i++)
{
     rwBuf.data[i] = ...;        // user data
}

result = devctl ( fd,                         // TDRV008 device handle
                  DCMD_TDRV008_WRITE,         // control code
                  &rwBuf,                     // I/O buffer
                  sizeof(rwBuf),
                  NULL);

...
```

```
...

if( result == EOK ) {
     // Process data, rwBuf.validWords is the real write result
     for (i = 0; i < rwBuf.validWords; i++)
     {
          printf(“@0x&04X: 0x%02X”, i, rwBuf.data[i]);
     }
     ...
}
else {
     // Process devctl() error
     ...
}
...
```

ERRORS

EFAULT	Invalid pointer to the user buffer.
EINVAL	An argument specified value is invalid.
ECHRNG	The port specified by I/O buffer member portNo doesn’t exist.
EIO	The port was configured as input port. It has to be an output port to start a write request.
EBUSY	The certain port is in use.

SEE ALSO

Library Reference - devctl()

3.3.3 DCMD_TDRV008_GETPORT

NAME

DCMD_TDRV008_GETPORT – Get the state of the 8-bit GPI port #4

DESCRIPTION

This TDRV008 control function reads the state of the free input lines of the 8 bit general purpose port 4. Only the upper 5 bits of the value are valid the lower 3 bits will always be set to 0. A pointer to a caller's output buffer (*unsigned char*) and the size of this structure are passed by the parameters *data_ptr* and *n_bytes* to the device.

Example

```
#include “tdrv008.h”
...
int      fd;
int      result;
unsigned char ucVal;

result = devctl(  fd,                        //  TDRV008 handle
                  DCMD_TDRV008_GETPORT,      //  control code
                  &ucVal,                    //  input buffer
                  sizeof(ucVal),
                  NULL);

/*  Check the result of the last device I/O control operation */
if( result == EOK )
{
     printf("OK\n");
     printf("    port4 (bit7..3): %02Xh\n", ucVal);
     ...
}
else
{
     // Process devctl() error
     ...
}
...
```

ERRORS

EFAULT	Invalid pointer to the user buffer.

SEE ALSO

Library Reference - devctl()

3.3.4 DCMD_TDRV008_SETPORT

NAME

DCMD_TDRV008_SETPORT – Set the state of the 8-bit GPO port #5

DESCRIPTION

This TDRV008 control function sets the state of the free output lines of the general purpose port 5. Only the upper 5 bits of the value are valid the lower 3 bits are ignored. A pointer to a callers input buffer (*unsigned char*) and the size of this structure are passed by the parameters *data_ptr* and *n_bytes* to the device.

Example

```
#include “tdrv008.h”
...
int       fd;
int       result;
unsigned char ucVal;

ucVal = 0x42;      // bits 0..2 are ignored -> so 0x40 will be written

result = devctl(  fd,               //  TDRV008 handle
                  DCMD_TDRV008_SETPORT,  //  control code
                  &ucVal,
                  sizeof(ucVal),
                  NULL);
/* Check the result of the last device I/O control operation */
if( result == EOK )
{
     printf("OK\n");
     ...
}
else
{
     // Process devctl() error
      ...
}
...
```

ERRORS

EFAULT	Invalid pointer to the user buffer.

SEE ALSO

Library Reference - devctl()

3.3.5 DCMD_TDRV008_CONFPORT

NAME

DCMD_TDRV008_CONFPORT – Port setup

DESCRIPTION

This TDRV008 control function configures a specified handshake port. A pointer to a callers configuration buffer (*TDRV008_CONF_BUFFER*) and the size of this structure are passed by the parameters *data_ptr* and *n_bytes* to the device.

The *TDRV008_CONF_BUFFER* structure has the following layout:

```
typedef struct
{
        int                 portNo;
        unsigned long       flags;
        int                 enaOutput;
        unsigned short      fifoTimeout;
        unsigned short      fifoThreshold;
} TDRV008_CONF_BUFFER, *PTDRV008_CONF_BUFFER;
```

Members

portNo

This parameter specifies the handshake port. Valid values are 0, 1 and 2.

flags

This parameter specifies the output handshake mode. (Refer to the User Manual of your module for a detailed description of the output handshake modes). Following values are valid:

TDRV008_F_CONF_HOUT_HSNONE	No output handshake
TDRV008_F_CONF_HOUT_HSINTERLOCKED	Interlocked output handshake
TDRV008_F_CONF_HOUT_HSPULSED	Pulsed output handshake

enaOutput

This parameter defines the direction of the port. If this parameter is set *TRUE* the port will be configured as an output port, if it is specified *FALSE* the port will be configured as input.

fifoTimeout

This parameter specifies the hardware FIFO timeout value. The value will be directly written to the module (Register TCPRx - refer to the User Manual of your module for more information). This value is only used for input ports.

fifoThreshold

This parameter specifies the FIFO threshold value. This value will be directly written to the module (Register FIFO_FTRx - refer to the User Manual of your module for more information). Valid values 1 to 512.

EXAMPLE

```
#include  “tdrv008.h”

...

int       fd;
int       result;
UCHAR ucVal;
TDRV008_CONF_BUFFER confBuf;

...

/* Setup handshake port 0 */
/* - output */
/* - interlocked output handshake */
/* - threshold: 256 */
confBuf.portNo =   0;
confBuf.flags =    TDRV008_F_CONF_HOUT_HSINTERLOCKED;
confBuf.enaOutput =     TRUE;
confBuf.fifoTimeout =   0;   /* not used */
confBuf.fifoThreshold = 256;

printf("\nConfigure port ... ");
result = devctl(  fd,                          //  TDRV008 handle
                  DCMD_TDRV008_CONFPORT,       //  control code
                  &confBuf,
                  sizeof(confBuf),
                  NULL);

...
```

```
...

//
//  Check the result of the last device I/O control operation
//
if( result == EOK )
{
     printf("OK\n");
}
else
{
     // Process devctl() error
}
...
```

ERRORS

EFAULT	Invalid pointer to the user buffer.
EINVAL	An argument specified value is invalid.
ECHRNG	The port specified by I/O buffer member portNo doesn't exist.
EIO	The port was configured as input port. It has to be an output port to start a write request.
EBUSY	The certain port is in use.

SEE ALSO

Library Reference - devctl()

3.3.6 DCMD_TDRV008_FLUSHPORTS

NAME

DCMD_TDRV008_FLUSHPORTS – Flush the hardware FIFOs of the 16-bit ports

DESCRIPTION

This TDRV008 control function flushes the FIFOs of all handshake ports (0, 1, and 2). This may be useful after configuration. The parameter pointer *data_ptr* and size *n_bytes* are not used for this devctl function.

EXAMPLE

```
#include “tdrv008.h“

...
int       fd;
int       result;

printf("\nFlush ports ... ");
result = devctl(  fd,                         //  TDRV008 handle
                  DCMD_TDRV008_FLUSHPORTS,    //  control code
                  NULL,
                  0,
                  NULL);
//
//  Check the result of the last device I/O control operation
//
if( result == EOK )
{
     printf("OK\n");
}
else
{
     // Process devctl() error
     ...
}
...
```

SEE ALSO

Library Reference - devctl()