The Embedded I/O Company

TEWS TECHNOLOGIES

TDRV008-SW-42

VxWorks Device Driver

3x16bit I/O Ports with 512 Word FIFO and Handshake

Version 1.0.x

User Manual

Issue 1.0.0

April 2005

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

TEWS TECHNOLOGIES LLC
1 E. Liberty Street, Sixth Floor
Reno, Nevada 89504 / USA
Phone: +1 (775) 686 6077
Fax: +1 (775) 686 6024
e-mail: usasales@tews.com
www.tews.com

TDRV008-SW-42

3x16bit I/O Ports
with 512 Word FIFO
and Handshake

VxWorks Device Driver

Supported Modules:
TPMC680-50
TPMC682

Issue	Description	Date
1.0.0	First Issue	April 11, 2005

Table of Content

1 Introduction

The TDRV008-SW-42 VxWorks device driver allows the operation of TDRV008 devices conforming to the VxWorks I/O system specification. This includes a device-independent basic I/O interface with *open()*, *close()*, *read()*, *write()*, and *ioctl()*.

TDRV008 will be used in the follow representative for all this modules and module names.

Supported Modules:

TPMC680-50 TPMC682	3x16 bit digital I/O with handshake and FIFO

The TDRV008 device driver includes the following functions:

- buffered reading and writing data to handshake ports
- configuration of handshake ports
- flushing all FIFOs of a module
- setting value of output port 5
- reading value from input port 4

2 Installation

Following files are located on the distribution media:

tdrv008drv.c	TDRV008 Driver Source
tdrv008exa.h	TDRV008 Device Configuration Table
tdrv008pci.c	PCI-Initialization for TDRV008 in x86-systems
tpxxxhwdep.c	Hardware dependent functions for TPMC modules
tdrv008.h	TDRV008 Driver Include File
tdrv008def.h	TDRV008 Internal Driver Include File
tpxxxhwdep.c	Hardware dependent functions Driver Include File
TDRV008-SW-42.pdf	This manual
Release.txt	Information about the Device Driver Release

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

2.1 Install the driver to the VxWorks-System

You have to perform the following steps to install the TDRV008 device driver to your VxWorks-System.

- Build the object code of the TDRV008 device driver.
- Link or load the driver object file to your VxWorks-System.
- Call the 'tdrv008Drv()' function to install the driver.

2.2 Including the driver in VxWorks

How to include the device driver in VxWorks systems is described in the VxWorks and Tornado manuals.

2.3 Special installation for Intel x86 based targets

The TDRV008 device driver is fully adapted for Intel x86 based targets. This is done by conditional compilation directives inside the source code and controlled by the VxWorks global defined macro ***CPU***. If the contents of this macro is equal to *I80386*, *I80386* or *PENTIUM* special Intel x86 conforming code and function calls will be included.

The second problem for Intel x86 based platforms can't be solved by conditional compilation directives. Due to the fact that some Intel x86 BSP's doesn't map PCI memory spaces of devices which are not used by the BSP, we can't access the required PCI memory spaces.

To solve this problem we have to add a MMU mapping entry for the required TDRV008 PCI memory spaces prior the MMU initialization (*usrMmuInit()*) is done.

Please examine the BSP documentation or contact the BSP Vendor whether the BSP perform automatic PCI and MMU configuration or not. If the PCI and MMU initialization is done by the BSP we won't include the function tdrv008PciInit() and can skip the following steps.

The C source file **tdrv008pci.c** contains the function *tdrv008PciInit().* This routine find out all TDRV008 devices and adds MMU mapping entries for all used PCI memory spaces. Please insert a call to this function after the PCI initialization is done and prior to MMU initialization (*usrMmuInit()*).

If you use the Tornado 2.0 project facility, the right place to call the function *tdrv008PciInit()* is at the end of the function *sysHwInit()* in **sysLib.c** (can be open from the project *Files* window).

If you use Tornado 1.0.1 compatibility tools insert the call to tdrv008PciInit() at the beginning of the root task (*usrRoot()*) in **usrConfig.c**.

Be sure that the function is called prior to MMU initialization otherwise the TDRV008 PCI spaces remains unmapped and we got an access fault during driver initialization.

Please insert the following call at a suitable place in either **sysLib.c** or **usrConfig.c**:

```
tdrv008PciInit();
```

To link the driver object modules to VxWorks, simply add all necessary driver files to the project. If you use Tornado 1.0.1 *Standard BSP Builds...* add the object modules to the macro *MACH_EXTRA* inside the BSP Makefile (MACH_EXTRA = tdrv008drv.o ...).

The Function tdrv008PciInit() was designed for and tested on generic Pentium targets. If you use another BSP please refer to BSP documentation or contact the technical support for required adaptation.

If you got strange errors after system startup with the new build system please carry out a VxWorks *build clean* and *build all*.

3 I/O system functions

This chapter describes the driver-level interface to the I/O system. The purpose of this function is to install the driver in the I/O system, add and initialize devices.

3.1 tdrv008Drv()

NAME

tdrv008Drv() - install the TDRV008 device driver in the I/O system and initialize the driver.

SYNOPSIS

STATUS tdrv008Drv(void)

DESCRIPTION

This function installs the TDRV008 device driver in the I/O system, allocates driver resources and initializes them. The call of this function is the first thing we have to do, before adding any device to the system or performing any I/O request.

RETURNS

OK or ERROR if the driver cannot be installed

SEE ALSO

VxWorks Programmer's Guide: I/O System

3.2 tdrv008DevCreate()

NAME

tdrv008DevCreate() - add a TDRV008 device to the system and initialize device hardware

SYNOPSIS

```
STATUS tdrv008DevCreate
(
    char        *name,      /* name of the device to create      */
    int         glbModNo    /* global module number to use       */
)
```

DESCRIPTION

This routine creates a TDRV008 device on a specified module that will be serviced by the TDRV008 device driver. This function must be called before performing any I/O request to this device.

The name of the device is selected by the string, which is deployed by this routine in the parameter **name**.

The argument glbModNo selects the module. (0, 1, …) How these numbers are assigned depends on the BSP and the order it detects the modules.

EXAMPLE

```
#include "tdrv008.h"

...

/*-------------------------------------------------------
  Create the device "/tdrv008A" for the first found module
  -------------------------------------------------------*/
status = tdrv008DevCreate (  "/tdrv008A", 0);

...
```

RETURNS

OK or ERROR if the driver is not installed or the device already exists or any other error occurred during the creation.

4 I/O interface functions

This chapter describes the interface to the basic I/O system.

4.1 open()

NAME

open() - open a device or file

SYNOPSIS

```
int open
(
        const char      *name,          /* name of the device to open                   */
        int             flags,          /* not used for TDRV008 driver, must be 0       */
        int             mode            /* not used for TDRV008 driver, must be 0       */
)
```

DESCRIPTION

Before I/O can be performed to the TDRV008 device, a file descriptor must be opened by invoking the basic I/O function *open()*.

EXAMPLE

```
...

/*----------------------------------------------
  open the device named "/tdrv008A" for I/O
  ----------------------------------------------*/
fd = open ("/tdrv008A", 0, 0);

...
```

RETURNS

A device descriptor number, or ERROR if the device does not exist or no device descriptors are available.

SEE ALSO

ioLib, basic I/O routine - open()

4.2 close()

NAME

close() – close a device or file

SYNOPSIS

```
int close
(
      int                 fd                  /* descriptor to close                  */
)
```

DESCRIPTION

This function closes opened devices.

EXAMPLE

```
int  retval;

...

/*------------------
  close the device
  ------------------*/
retval = close(fd);

...
```

RETURNS

Returns OK if device has been closed successfully, or ERROR if the descriptor is invalid.

SEE ALSO

ioLib, basic I/O routine - close()

4.3 read()

NAME

read() – receives data from read FIFO from the specified handshake port.

SYNOPSIS

```
int read
(
        int         fd,             /* device descriptor from opened TDRV008 device   */
        char        *buffer,        /* pointer to the read structure                  */
        size_t      nbytes          /* not used                                       */
)
```

DESCRIPTION

This function receives data from a specified handshake port FIFO and copies the data into the buffer.

It is necessary that the specified handshake port is configured for input.

PARAMETER

fd

This parameter is a file descriptor specifying the device which shall be used.

buffer

This parameter points to a data structure *TDRV008_RW_BUFFER* (see below).

nbytes

This argument is not used for the device driver.

data structure *TDRV008_RW_BUFFER*:

```
typedef struct tdrv008_rw_buffer
{
        int             portNo;
        unsigned long   flags;
        int             timeout;
        int             bufferSize;
        int             validWords;
        unsigned short  *buffer;
} TDRV008_RW_BUFFER, *PTDRV008_RW_BUFFER;
```

portNo

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

flags

This parameter is an ORed value of the following flags.

Flag	Description
TDRV008_F_RW_NOWAIT	Read only the data actually in the FIFO, do not wait in any way.

timeout

This parameter specifies the maximum time the function should block before it returns.

bufferSize

This value specifies the size of the buffer in words (16-bit). The function will try to place the specified number of data.

validWords

The argument returns the number of received data words.

buffer

This parameter is a pointer to the application supplied input data buffer.

EXAMPLE

```
#include “tdrv008.h”

…

#define             BUF_SIZE                20

int                 retval;
int                 dev;
TDRV008_RW_BUFFER   rwBuf;
unsigned short      rcvBuffer[BUF_SIZE] =   {…};

…

dev = open(…);

…
```

…

```
/* Read out FIFO of portNo 2, return immediately */
rwBuf.portNo =     1;
rwBuf.flags =      TDRV008_F_RW_NOWAIT;
rwBuf.bufferSize = BUF_SIZE;
rwBuf.bufferSize = 0;
rwBuf.buffer =     trmBuffer;
rwBuf.timeout =    0;                /* not used */

retval = read(tdrv008_dev[device], (void*)&rwBuf, 0);
if (retval == ERROR)
{
     /* Error occurred */
}
else
{
     /* All data sent */
}
```

…

RETURNS

ERROR or number of bytes received

INCLUDE FILES

tdrv008.h

SEE ALSO

ioLib, basic I/O routine - *read()*

4.4 write()

NAME

write() – copies content of a data buffer to the specified handshake port.

SYNOPSIS

```
int write
(
        int         fd,             /* device descriptor from opened TDRV008 device   */
        char        *buffer,        /* pointer to the write structure                 */
        size_t      nbytes          /* not used                                       */
)
```

DESCRIPTION

This function transmits the data placed in the buffer via the specified handshake port.

It is necessary that the specified handshake port is configured for output.

PARAMETER

fd

This parameter is a file descriptor specifying the device which shall be used.

buffer

This parameter points to a data structure *TDRV008_RW_BUFFER* (see below).

nbytes

This argument is not used for the device driver.

data structure *TDRV008_RW_BUFFER*:

```
typedef struct tdrv008_rw_buffer
{
        int             portNo;
        unsigned long   flags;
        int             timeout;
        int             bufferSize;
        int             validWords;
        unsigned short  *buffer;
} TDRV008_RW_BUFFER, *PTDRV008_RW_BUFFER;
```

portNo

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

flags

This is not used for write().

timeout

This parameter specifies the maximum time the function should block before it returns. If data could be written to the output FIFO before this time expires, the function will return earlier. The time is specified in ticks. A value of -1 specifies that the function should not timeout. If function times out an error (S_tdrv008Drv_SENDTIMEOUT) is indicated, but the returned value of *validWords* will return the number of transmitted data words.

bufferSize

This value specifies the size of the buffer in words (16-bit). The function will send this amount of data.

validWords

The argument returns the number of transmitted data words. If transmission is blocked, the value may be smaller than *bufferSize*. If the complete buffer has been copied into the FIFO, the value will be bufferSize.

buffer

This parameter is a pointer to the application filled output data buffer.

EXAMPLE

```
#include “tdrv008.h”

…

#define             BUF_SIZE                20

int                 retval;
int                 dev;
TDRV008_RW_BUFFER   rwBuf;
unsigned short      trmBuffer[BUF_SIZE] =   {…};

…

dev = open(…);

…
```

…

```
/* Transmit buffer on portNo 1, timeout after 120 ticks */
rwBuf.portNo =     1;
rwBuf.flags =      0;
rwBuf.bufferSize = BUF_SIZE;
rwBuf.bufferSize = 0;
rwBuf.buffer =     trmBuffer;
rwBuf.timeout =    120;

retval = write(tdrv008_dev[device], (void*)&rwBuf, 0);
if (retval == ERROR)
{
     /* Error occurred */
     if (errnoGet() == S_tdrv008Drv_SENDTIMEOUT)
     {
          printf("%d words have been sent\n", rwBuf.validWords);
     }
}
else
{
     /* All data sent */
}
```

…

RETURNS

ERROR or number of bytes written

INCLUDE FILES

tdrv008.h

SEE ALSO

ioLib, basic I/O routine - *write()*

4.5 ioctl()

NAME

ioctl() - performs an I/O control function.

SYNOPSIS

```
int ioctl
(
    int     fd,         /* device descriptor from opened TDRV008 device  */
    char    request,    /* select of control function                   */
    int     arg         /* parameter buffer                              */
)
```

DESCRIPTION

Special I/O operation that do not fit to the standard basic I/O calls will be performed by calling the *ioctl()* function.

PARAMETER

fd

This parameter is a file descriptor specifying the device which shall be used.

request

This parameter specifies the function that shall be executed.

arg

This argument is a argument special to the request.

RETURNS

OK (if the function completes successfully) or ERROR

INCULDES

tdrv008.h

SEE ALSO

ioLib, basic I/O routine - *ioctl()*

4.5.1 FIO_TDRV008_GETPORT Read state of input port 4

This I/O control function reads the actual state of the free input lines on port 4. The function dependent argument **arg** points to an application supplied unsigned character which is used to return the input value. Only the upper 5 bit of the value are valid the lower 3 bits will always be set to 0.

EXAMPLE

```
#include “tdrv008.h”

…

int                retval;
int                dev;
unsigned char      port4Val;

…

dev = open(…);

…

/* Read input state of port 4 */
retval = ioctl(dev, FIO_TDRV008_GETPORT, (int)&port4Val);
if (retval == ERROR)
{
     /* Error occurred */
}
else
{
     /* OK */
}

…
```

4.5.2 FIO_TDRV008_SETPORT Set value of output port 5

This I/O control function sets the state of the free output lines on port 5. The function dependent argument **arg** points to an application supplied unsigned character value which specifies the output value. Only the upper 5 bit of the value are valid the lower 3 bits are ignored.

EXAMPLE

```
#include “tdrv008.h”

…

int                retval;
int                dev;

…

dev = open(…);

…

/* Set output lines port5 line 3..7 (set line 3,5,7) */
retval = ioctl(dev, FIO_TDRV008_GETPORT, 0xA8);
if (retval == ERROR)
{
     /* Error occurred */
}
else
{
     /* OK */
}

…
```

4.5.3 FIO_TDRV008_CONFPORT Configure handshake port

This I/O control function configures a specified handshake port. The function dependent argument **arg** points to a structure *TDRV008_CONF_BUFFER*..

data structure *TDRV008_CONF_BUFFER:*

```
typedef struct tdrv008_conf_buffer
{
        int             portNo;
        unsigned long   flags;
        int             enaOutput;
        unsigned short  fifoTimeout;
        unsigned short  fifoThreshold;
} TDRV008_CONF_BUFFER, *PTDRV008_CONF_BUFFER;
```

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.

Flag	**Description**
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 (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 (FIFO_FTRx). (Refer to the User Manual of your module for more information). This value must be set between 1 and 512.

EXAMPLE

```
#include “tdrv008.h”

…

int                     retval;
int                     dev;
TDRV008_CONF_BUFFER     confBuf;

…

dev = open(…);

…

/* Setup handshake port 0 */
/* - output */
/* - interlocked output handshake */
/* - threashold: 256 */
confBuf.portNo =        0;
confBuf.flags =         TDRV008_F_CONF_HOUT_HSINTERLOCKED;
confBuf.enaOutput =     TRUE;
confBuf.fifoTimeout =   0;      /* not used */
confBuf.fifoThreshold = 256;
retval = ioctl(dev, FIO_TDRV008_CONFPORT, (int)&confBuf);
if (retval == ERROR)
{
     /* Error occurred */
}
else
{
     /* OK */
}

…
```

4.5.4 FIO_TDRV008_FLUSHPORTS Flush FIFOs of all handshake ports

This I/O control function flushes the FIFOs of all handshake ports (0, 1, and 2). This may be useful after configuration. The function dependent argument **arg** is not used for this function.

EXAMPLE

```
#include “tdrv008.h”

…

int                retval;
int                dev;

…

dev = open(…);

…

/* Flush FIFOs */
retval = ioctl(dev, FIO_TDRV008_FLUSHPORTS, 0);
if (retval == ERROR)
{
     /* Error occurred */
}
else
{
     /* OK */
}

…
```

5 Appendix

5.1 Error Codes

If the device driver creates an error the error codes are stored in the *errno*. They can be read with the VxWorks function *errnoGet()* or *printErrno().*

S_tdrv008Drv_NXIO	0xE0080000	No TDRV008 device found on the specified place
S_tdrv008Drv_NODRV	0xE0080001	No TDRV008 driver installed
S_tdrv008Drv_NODEV	0xE0080002	Can not find the specified TDRV008 device. (Illegal module number)
S_tdrv008Drv_EXISTS	0xE0080003	The specified device has already been created
S_tdrv008Drv_BUSY	0xE0080004	The driver is still busy and can not be uninstalled
S_tdrv008Drv_NOMEM	0xE0080005	Can not allocate memory
S_tdrv008Drv_NOBUF	0xE0080006	There is no parameter buffer specified
S_tdrv008Drv_INVPORT	0xE0080007	An invalid port number has been specified
S_tdrv008Drv_UNSUPFLAG	0xE0080008	The flags are not valid or the combination is invalid
S_tdrv008Drv_PORTBUSY	0xE0080009	The specified port is busy, try later
S_tdrv008Drv_INVDIRECTION	0xE008000A	The read request has been called for an output port or a write request has been called for an input port
S_tdrv008Drv_ICMD	0xE008000B	An unsupported ioctl command has been specified
S_tdrv008Drv_INVPARA	0xE008000C	An parameter value is invalid
S_tdrv008Drv_SENDTIMEOUT	0xE008000D	The write request has timed out without sending the complete data buffer