The Embedded I/O Company

TEWS TECHNOLOGIES

TDRV003-SW-72

LynxOS Device Driver

16 (8) Digital Inputs

16 (8) Digital Outputs

Version 1.0.x

User Manual

Issue 1.0.0

November 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

TDRV003-SW-72

LynxOS Device Driver

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

Supported Modules:
TPMC670
TPMC671

Issue	Description	Date
1.0.0	First Issue	November 6, 2009

Table of Contents

1 INTRODUCTION 4

2 INSTALLATION 5

2.1 Device Driver Installation 6

2.1.1 Static Installation 6

2.1.1.1 Build the driver object 6

2.1.1.2 Create Device Information Declaration 6

2.1.1.3 Modify the Device and Driver Configuration File 6

2.1.1.4 Rebuild the Kernel 7

2.1.2 Dynamic Installation 7

2.1.2.1 Build the driver object 7

2.1.2.2 Create Device Information Declaration 7

2.1.2.3 Uninstall dynamic loaded driver 8

2.1.3 Device Information Definition File 8

2.1.4 Configuration File: CONFIG.TBL 9

3 TDRV003 DEVICE DRIVER PROGRAMMING 10

3.1 open() 10

3.2 close() 12

3.3 ioctl() 13

3.3.1 TDRV003_READ 14

3.3.2 TDRV003_WRITE 17

3.3.3 TDRV003_DEBENABLE 19

3.3.4 TDRV003_DEBDISABLE 20

3.3.5 TDRV003_WDENABLE 21

3.3.6 TDRV003_WDDISABLE 22

3.3.7 TDRV003_WDRESET 23

4 DEBUGGING AND DIAGNOSTIC 24

1 Introduction

The TDRV003-SW-72 LynxOS device driver allows the operation of the TDRV003 digital I/O PMC on LynxOS platforms with DRM based PCI interface.

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

The TDRV003-SW-72 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-72 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)

To get more information about the features and use of TDRV003 devices it is recommended to read the manuals listed below.

TPMC670 User Manual

TPMC671 Engineering Manual

2 Installation

Following files are located on the distribution media:

Directory path 'TDRV003-SW-72':

TDRV003-SW-72-SRC.tar.gz	GZIP compressed archive with driver source code
TDRV003-SW-72-1.0.0.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 TDRV003-SW-72-SRC.tar.gz contains the following files and directories:

Directory path 'tdrv003':

tdrv003.c	TDRV003 device driver source
tdrv003def.h	TDRV003 driver include file
tdrv003.h	TDRV003 include file for driver and application
tdrv003_info.c	TDRV003 Device information definition
tdrv003_info.h	TDRV003 Device information definition header
tdrv003.cfg	TDRV003 Driver configuration file include
tdrv003.import	Linker import file
Makefile	Device driver make file
example/ tdrv003exa.c	Example application
example/Makefile	Example application makefile

In order to perform a driver installation, first extract the TAR file to a temporary directory, than follow the steps below:

1. Create a new directory in the system drivers directory path /sys/drivers.xxx, where xxx represents the BSP that supports the target hardware.

   For example: /sys/drivers.pp_drm/tdrv003 or /sys/drivers.cpci_x86/tdrv003

2. Copy the following files to this directory:
   - tdrv003.c
   - tdrv003def.h
   - tdrv003.import
   - Makefile

3. Copy tdrv003.h to /usr/include/

4. Copy tdrv003_info.c to /sys/devices.xxx/ or /sys/devices if /sys/devices.xxx does not exist (xxx represents the BSP).

5. Copy tdrv003_info.h to /sys/dheaders/

Copy tdtv003.cfg to */sys/cfg.xxx/*, where xxx represents the BSP for the target platform. For example: /sys/cfg.ppc or /sys/cfg.x86 ….

2.1 Device Driver Installation

The two methods of driver installation are as follows:

(1) Static Installation
(2) Dynamic Installation (only native LynxOS 4 systems)

2.1.1 Static Installation

With this method, the driver object code is linked with the kernel routines and is installed during system start-up.

2.1.1.1 Build the driver object

1. Change to the directory /sys/drivers.xxx/tdrv003, where xxx represents the BSP that supports the target hardware.
2. To update the library /sys/lib/libdrivers.a enter:

   ```
   make install
   ```

2.1.1.2 Create Device Information Declaration

1. Change to the directory /sys/devices.xxx/ or /sys/devices if /sys/devices.xxx does not exist (xxx represents the BSP).
2. Add the following dependencies to the Makefile

   ```
   DEVICE_FILES_all = … tdrv003_info.x
   ```

   And at the end of the Makefile

   ```
   tdrv003_info.o:$(DHEADERS)/tdrv003_info.h
   ```

3. To update the library /sys/lib/libdevices.a enter:

   ```
   make install
   ```

2.1.1.3 Modify the Device and Driver Configuration File

In order to insert the driver object code into the kernel image, an appropriate entry in file CONFIG.TBL must be created.

1. Change to the directory /sys/lynx.os/ respective /sys/bsp.xxx, where xxx represents the BSP that supports the target hardware.
2. Create an entry at the end of the file CONFIG.TBL

   Insert the following entry at the end of this file.

   ```
   I:tdrv003.cfg
   ```

2.1.1.4 Rebuild the Kernel

1. Change to the directory /sys/lynx.os/ (/sys/bsp.xxx)
2. Enter the following command to rebuild the kernel:

   ```
   make install
   ```

3. Reboot the newly created operating system by the following command (not necessary for KDIs):

   ```
   reboot –aN
   ```

   The N flag instructs init to run mknod and create all the nodes mentioned in the new nodetab.

4. After reboot you should find the following new devices (depends on the device configuration): /dev/tdrv003a, /dev/tdrv003b, ….

2.1.2 Dynamic Installation

This method allows you to install the driver after the operating system is booted. The driver object code is attached to the end of the kernel image and the operating system dynamically adds this driver to its internal structures. The driver can also be removed dynamically.

2.1.2.1 Build the driver object

1. Change to the directory /sys/drivers.xxx/tdrv003, where xxx represents the BSP that supports the target hardware.
2. To make the dynamic link-able driver enter:

   ```
   make dldd
   ```

2.1.2.2 Create Device Information Declaration

1. Change to the directory /sys/drivers.xxx/tdrv003, where xxx represents the BSP that supports the target hardware.
2. To create a device definition file for the major device (this works only on native systems)

   ```
   make td003info
   ```

3. To install the driver enter:

   ```
   drinstall –c tdrv003.obj
   ```

   If successful, drinstall returns a unique <driver-ID>

4. To install the major device enter:

   ```
   devinstall –c –d <driver-ID> td003info
   ```

   The <driver-ID> is returned by the drinstall command

5. To create the node for the devices enter:

   ```
   mknod /dev/tdrv003a c <major_no> 0
   ```

   The <major_no> is returned by the devinstall command.

If all steps are successfully completed, the TDRV003 is ready to use.

2.1.2.3 Uninstall dynamic loaded driver

To uninstall the TDRV003 device enter the following commands:

```
devinstall –u –c <device-ID>
drinstall –u <driver-ID>
```

2.1.3 Device Information Definition File

The device information definition contains information necessary to install the TDRV003 major device.

The implementation of the device information definition is done through a C structure, which is defined in the header file *tdrv003_info.h*.

This structure contains the following parameter:

PCIBusNumber Contains the PCI bus number at which the supported device is connected. Valid bus numbers are in range from 0 to 255.

PCIDeviceNumber Contains the device number (slot) at which the supported device is connected. Valid device numbers are in range from 0 to 31.

If both PCIBusNumber and PCIDeviceNumber are –1 then the driver will auto scan for supported devices. The first device found in the scan order will be allocated by the driver for this major device.

Already allocated devices can't be allocated twice. This is important to know if there are more than one TDRV003 major devices.

maxNumJob Specifies the number of event jobs for the device that will be able to wait at the same time. The value must be greater than 0.

A device information definition is unique for every TDRV003 major device. The file *tdrv003_info.c* on the distribution media contains two device information declarations, **tdrv003A** for the first major device and **tdrv003B** for the second major device.

If the driver should support more than two major devices it is necessary to copy and paste an existing declaration and rename it with a unique name, for example **tdrv002C**, **tdrv003D** and so on.

It is also necessary to modify the device and driver configuration file, respectively the configuration include file *tdrv003.cfg*.

The following device declaration information uses the auto find method to detect a supported device on the PCI bus.

```
TDRV003_INFO tdrv003A = {

    -1,           /*  Auto find the device on any PCI bus    */
    -1,
    5             /* max. number of pending jobs */
};
```

2.1.4 Configuration File: CONFIG.TBL

The device and driver configuration file CONFIG.TBL (respective config.tbl on LynxOS 5.0 systems) contains entries for device drivers and its major and minor device declarations. Each time the system is rebuild, the config utility read this file and produces a new set of driver and device configuration tables and a corresponding nodetab.

To install the TDRV003 driver and devices into the LynxOS system, the configuration include file tdrv003.cfg must be included in the CONFIG.TBL (see also chapter 2.1.1.3).

The file tdrv003.cfg on the distribution disk contains the driver entry (*C:tdrv003:*\....) and two major device entries (*D:TDRV003 1:tdrv003A:: and D:TDRV003 2:tdrv003B::*).

If the driver should support more than one major device, the following entries for major devices must be enabled by removing the comment character (#). By copy and paste an existing major and minor entries and renaming the new entries, it is possible to add any number of additional TDRV003 devices.

This example shows a driver entry with two major devices and one minor device:

```
#    Format:
#    C:driver-name:open:close:read:write:select:control:install:uninstall
#    D:device-name:info-block-name:raw-partner-name
#    N:node-name:minor-dev

C:tdrv003:tdrv003open:tdrv003close: \
::: \
::tdrv003ioctl: \
:tdrv003install:tdrv003uninstall
D:TDRV003 1:tdrv003A::
N:tdrv003a:0
D:TDRV003 2:tdrv003B::
N:tdrv003b:0
```

The configuration above creates the following nodes in the /dev directory.

```
/dev/tdrv003a /dev/tdrv003b
```

3 TDRV003 Device Driver Programming

LynxOS system calls are all available directly to any C program. They are implemented as ordinary function calls to "glue" routines in the system library, which trap to the OS code.

Note that many system calls use data structures, which should be obtained in a program from appropriate header files. Necessary header files are listed with the system call synopsis.

3.1 open()

NAME

open() - open a file

SYNOPSIS

#include <sys/file.h>
#include <sys/types.h>
#include <fcntl.h>

int open (char *path, int oflags[, mode_t mode])

DESCRIPTION

Opens a file (TDRV003 device) named in ***path*** for reading and writing. The value of ***oflags*** indicates the intended use of the file. In case of a TDRV003 device ***oflags*** must be set to **O_RDWR** to open the file for both reading and writing.

The ***mode*** argument is required only when a file is created. Because a TDRV003 device already exists this argument is ignored.

EXAMPLE

```
int fd;

fd = open ("/dev/tdrv003a", O_RDWR);
if (fd == -1)
{
     /* Handle error */
}
```

RETURNS

open returns a file descriptor number if successful, or –1 on error.

SEE ALSO

LynxOS System Call - open()

3.2 close()

NAME

close() – close a file

SYNOPSIS

int close(int fd)

DESCRIPTION

This function closes an opened device.

EXAMPLE

```
int  result;

result = close(fd);
if (result == -1)
{
     /* Handle error */
}
```

RETURNS

close returns 0 (OK) if successful, or –1 on error

SEE ALSO

LynxOS System Call - close()

3.3 ioctl()

NAME

ioctl() – I/O device control

SYNOPSIS

#include <ioctl.h>
#include <tdrv003.h>

int ioctl (int fd, int request, char *arg)

DESCRIPTION

ioctl provides a way of sending special commands to a device driver. The call sends the value of request and the pointer arg to the device associated with the descriptor fd.

The following ioctl codes are supported by the driver and are defined in *tdrv003.h*:

Symbol	Meaning
TDRV003_READ	Read input state (on event)
TDRV003_WRITE	Set output lines
TDRV003_DEBENABLE	Setup and enable input debouncing
TDRV003_DEBDISABLE	Disable input debouncing
TDRV003_WDENABLE	Enable output watchdog
TDRV003_WDDISABLE	Disable output watchdog
TDRV003_WDRESET	Delete watchdog error

See behind for more detailed information on each control code.

RETURNS

ioctl returns 0 if successful, or –1 on error.

On error, *errno* will contain a standard error code (see also LynxOS System Call – ioctl).

SEE ALSO

LynxOS System Call - ioctl().

3.3.1 TDRV003_READ

NAME

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 parameter *arg* 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 ticks) 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_BUFFER     rdBuf;

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

…
```

```
…

result = ioctl(fd, TDRV003_READ, (char*)&rdBuf);
if (result >= 0)
{
     /* 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      = 10000;           /* 10000 ticks */

result = ioctl(fd, TDRV003_READ, (char*)&rdBuf);
if (result >= 0)
{
     /* Read has been successful */
     printf("Input State: %04Xh\n", rdBuf.value);
}
else
{
     /* Read failed */
}
```

ERRORS

EINVAL	An unsupported input parameter value has been specified. Check input parameters
ENOSPC	The maximum number of pending jobs is already pending. (Increase number of maximum jobs in tdrv003_info.h)
EINTR	The function was cancelled.
ETIMEDOUT	The wait time has exceeded the maximum time of a sequencer cycle. (no interrupts or HW-problem)

Other returned error codes are system error conditions.

3.3.2 TDRV003_WRITE

NAME

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 parameter *arg* 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 = ioctl(fd, TDRV003_WRITE, (char*)&wrBuf);
if (result >= 0)
{
     /* Write has been successful */
}
else
{
     /* Write failed */
}
```

ERRORS

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

Other returned error codes are system error conditions.

3.3.3 TDRV003_DEBENABLE

NAME

TDRV003_DEBENABLE – Setup and enable input debouncing

DESCRIPTION

This function sets up the debouncer time and enables the input debouncer. The function dependent parameter *arg* points to a buffer (unsigned short) which specifies 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 = ioctl(fd, TDRV003_DEBENABLE, (char*)&debVal);
if (result >= 0)
{
     /* Debouuncer successfully started */
}
else
{
     /* Debouncer start failed */
}
```

3.3.4 TDRV003_DEBDISABLE

NAME

TDRV003_DEBDISABLE – Disable input debouncing

DESCRIPTION

This function disables the input debouncer. The function dependent parameter *arg* is not used and should be set to *NULL*.

EXAMPLE

```
#include <tdrv003.h>

int  fd;
int  result;

/* --- stop debouncinging --- */
result = ioctl(fd, TDRV003_DEBDISABLE, NULL);
if (result >= 0)
{
     /* Debouuncer successfully stopped */
}
else
{
     /* Debouncer stopped failed */
}
```

3.3.5 TDRV003_WDENABLE

NAME

TDRV003_WDENABLE – Enable output watchdog

DESCRIPTION

This function enables the output watchdog. The function dependent parameter *arg* is not used and should be set to *NULL*.

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 TDRV003_WD_RESET 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 = ioctl(fd, TDRV003_WDENABLE, NULL);
if (result >= 0)
{
     /* Output watchdog successfully enabled*/
}
else
{
     /* Enable watchdog failed */
}
```

3.3.6 TDRV003_WDDISABLE

NAME

TDRV003_WDDISNABLE – Disable output watchdog

DESCRIPTION

This function disables the output watchdog. The function dependent parameter *arg* is not used and should be set to *NULL*.

EXAMPLE

```
#include <tdrv003.h>

int  fd;
int  result;

/* --- disable output watchdog --- */
result = ioctl(fd, TDRV003_WDDISABLE, NULL);
if (result >= 0)
{
     /* Output watchdog successfully disabled*/
}
else
{
     /* Disable watchdog failed */
}
```

3.3.7 TDRV003_WDRESET

NAME

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 parameter *arg* is not used and should be set to *NULL*.

EXAMPLE

```
#include <tdrv003.h>

int  fd;
int  result;

/* --- reset output watchdog error --- */
result = ioctl(fd, TDRV003_WDRESET, NULL);
if (result >= 0)
{
     /* Status of watchdog successfully cleared */
}
else
{
     /* Clear of watchdog status failed */
}
```

4 Debugging and Diagnostic

If the driver will not work properly, please enable debug outputs by defining the symbols *DEBUG, DEBUG_TPMC,* and *DEBUG_PCI* in file tdrv003.c.

The debug output should appear on the console. If not, please check the symbol *KKPF_PORT* in *uparam.h*. This symbol should be configured to a valid COM port (e.g. *SKDB_COM1*).

The debug output displays the device information data for the current major device like this. The example shows the output for a TPMC670.

```
TEWS TECHNOLOGIES -TDRV003: Device Driver Install
Bus = 0  Dev = 17  Func = 0
[00] = 905010B5
[04] = 02800000
[08] = 11800000
[0C] = 00000000
[10] = 80004000
[14] = 00804001
[18] = 00805001
[1C] = 00000000
[20] = 00000000
[24] = 00000000
[28] = 00000000
[2C] = 029E1498
[30] = 00000000
[34] = 00000040
[38] = 00000000
[3C] = 0000010C

PCI Base Address 0 (PCI_RESID_BAR0)

70604000 : F1 FF FF 0F 00 00 00 00 00 00 00 00 00 00 00 00
70604010 : 00 00 00 00 01 00 00 00 00 00 00 00 00 00 00 00
70604020 : 00 00 00 00 00 00 00 00 A0 78 71 01 00 00 00 00
70604030 : 00 00 00 00 00 00 00 00 00 00 00 00 09 00 00 00

PCI Base Address 1 (PCI_RESID_BAR1)

PCI Base Address 2 (PCI_RESID_BAR2)

70205000 : FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
```

The debug output above is only an example. Debug output on other systems may be different for addresses and data in some locations.