*The Embedded I/O Company*

TEWS TECHNOLOGIES

# TDRV006-SW-72

## LynxOS Device Driver

64 Digital Inputs/Outputs (Bit I/O)

Version 1.0.x

## User Manual

Issue 1.0.0

January 2008

**TEWS TECHNOLOGIES GmbH**
Am Bahnhof 7
25469 Halstenbek, Germany
www.tews.com

Phone: +49 (0) 4101 4058 0
Fax: +49 (0) 4101 4058 19
e-mail: info@tews.com

**TEWS TECHNOLOGIES LLC**
9190 Double Diamond Parkway,
Suite 127, Reno, NV  89521, USA
www.tews.com

Phone: +1 (775) 850 5830
Fax: +1 (775) 201 0347
e-mail: usasales@tews.com

# TDRV006-SW-72

LynxOS Device Driver

64 Digital Inputs/Outputs (Bit I/O)

Supported Modules:
TPMC681

| Issue | Description | Date |
|---|---|---|
| 1.0.0 | First Issue | January 29, 2008 |

# 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 ........ 8

2.1.2.1 Build the driver object ........ 8

2.1.2.2 Create Device Information Declaration ........ 8

2.1.2.3 Uninstall dynamic loaded driver ........ 8

2.1.3 Device Information Definition File ........ 9

2.1.4 Configuration File: CONFIG.TBL ........ 10

3 TDRV006 DEVICE DRIVER PROGRAMMING ........ 11

3.1 open() ........ 11

3.2 close() ........ 13

3.3 ioctl() ........ 14

3.3.1 TDRV006_READ ........ 15

3.3.2 TDRV006_WRITE ........ 16

3.3.3 TDRV006_OUTPUT_ENABLE ........ 17

3.3.4 TDRV006_EVENT_WAIT ........ 18

4 DEBUGGING AND DIAGNOSTIC ........ 20

# 1 Introduction

The TDRV006-SW-72 LynxOS device driver allows the operation of the TPMC681 product family 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 TDRV006 device driver includes the following functions:

- configure direction of I/O lines
- set output value of output lines
- read value of I/O lines
- wait for I/O line events

The TDRV006-SW-72 device driver supports the modules listed below:

| | | |
|---|---|---|
| TPMC681 | 64 Digital Inputs/Outputs (Bit I/O) | (PMC) |

**In this document all supported modules and devices will be called TDRV006. Specials for certain devices will be advised.**

To get more information about the features and use of TDRV006 devices it is recommended to read the manuals listed below.

TPMC681 User manual

TPMC681 Engineering Manual

# 2 Installation

Following files are located on the distribution media:

Directory path '.\TDRV006-SW-72\':

| | |
|---|---|
| TDRV006-SW-72-1.0.0.pdf | This manual in PDF format |
| TDRV006-SW-72-SRC.tar | Device Driver and Example sources |
| Release.txt | Information about the Device Driver Release |
| ChangeLog.txt | Release history |

The TAR archive TDRV006-SW-72-SRC.tar contains the following files and directories:

| | |
|---|---|
| tdrv006.c | Driver source code |
| tdrv006.h | Definitions and data structures for driver and application |
| tdrv006def.h | Definitions and data structures for the driver |
| tdrv006_info.c | Device information definition |
| tdrv006_info.h | Device information definition header |
| tdrv006.cfg | Driver configuration file include |
| tdrv006.import | Linker import file |
| Makefile | Device driver make file |
| example/tdrv006exa.c | Example application source |
| example/Makefile | Example application make file |

In order to perform an installation, extract all files of the archive TDRV006-SW-72-SRC.tar.gz to the desired target directory. The command 'tar -xzvf TDRV006-SW-72-SRC.tar.gz' will extract the files into the local directory. Then 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/tdrv006 or /sys/drivers.cpci_x86/tdrv006

2. Copy the following files to this directory:
   - tdrv006.c
   - tdrv006def.h
   - tdrv006.import
   - Makefile

3. Copy tdrv006.h to /usr/include/

4. Copy tdrv006_info.c to /sys/devices.xxx/ or /sys/devices if /sys/devices.xxx does not exist (xxx represents the BSP).

5. Copy tdrv006_info.h to /sys/dheaders/

6. Copy tdrv006.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:

- Static Installation
- Dynamic Installation (only native LynxOS 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/tdrv006, 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 = ... tdrv006_info.x
   ```

   And at the end of the Makefile

   ```
   tdrv006_info.o:$(DHEADERS)/tdrv006_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:tdrv006.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/tdrv006a, /dev/tdrv006b, …

### 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/tdrv006, 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/tdrv006, 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 t006info
   ```

3. To install the driver enter:

   ```
   drinstall –c tdrv006.obj
   ```

   If successful, drinstall returns a unique <driver-ID>
4. To install the major device enter:

   ```
   devinstall –c –d <driver-ID> t006info
   ```

   The <driver-ID> is returned by the drinstall command
5. To create nodes for both minor devices enter:

   ```
   mknod /dev/tdrv006a c <major_no> 0
   ```

   The <major_no> is returned by the devinstall command.

If all steps are successfully completed, the TDRV006 device is ready to use.

#### 2.1.2.3 Uninstall dynamic loaded driver

To uninstall the TDRV006 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 TDRV006 major device.

The implementation of the device information definition is done through a C structure, which is defined in the header file *tdrv006_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 TDRV006 major devices.**

A device information definition is unique for every TDRV006 major device. The file *tdrv006_info.c* on the distribution media contains two device information declarations, **tdrv006a_info** for the first major device and **tdrv006b_info** 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 **tdrv006c_info**, **tdrv006d_info** and so on.

**It is also necessary to modify the device and driver configuration file, respectively the configuration include file *tdrv006.cfg*.**

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

```
TDRV006_INFO tdrv006a_info = {

     -1,           /*  Auto find the device on any PCI bus    */
     -1,

};
```

## 2.1.4 Configuration File: CONFIG.TBL

The device and driver configuration file CONFIG.TBL 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 TDRV006 driver and devices into the LynxOS system, the configuration include file tdrv006.cfg must be included in the CONFIG.TBL (see also chapter 2.1.1.3).

The file tdrv006.cfg on the distribution disk contains the driver entry (*C:tdrv006:\....*) and two major device entries ( *D:TDRV006 1:tdrv006a_info:: and D:TDRV006 2:tdrv006b_info::* ).

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 TDRV006 devices.

This example shows a driver entry with one major device 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:tdrv006:\
      :tdrv006open:tdrv006close:::\
      ::tdrv006ioctl:tdrv006install:tdrv006uninstall
D:TDRV006 1:tdrv006a_info::
N:tdrv006a:0
D:TDRV006 2:tdrv006b_info::
N:tdrv006b:0
```

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

```
/dev/tdrv006a  /dev/tdrv006b
```

# 3 TDRV006 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 (TDRV006 device) named in ***path*** for reading and writing. The value of ***oflags*** indicates the intended use of the file. In case of a TDRV006 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 TDRV006 device already exists this argument is ignored.

### EXAMPLE

```
int fd


/* open the device named "/dev/tdrv006a" for I/O */
fd = open ("/dev/tdrv006a", O_RDWR);
if (!fd)
{
     /* 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;

/*
**   close the device
*/
result = close(fd);
if (result < 0)
{
     /* 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 <tdrv006.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 *tdrv006.h*:

| **Symbol** | **Meaning** |
|---|---|
| TDRV006_READ | Read value of I/O lines |
| TDRV006_WRITE | Write value of output lines |
| TDRV006_OUTPUT_ENABLE | Enable output |
| TDRV006_EVENT_WAIT | Wait for I/O transition events |

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().

tdrv006exa.c programming example

## 3.3.1 TDRV006_READ

### NAME

TDRV006_READ – Read value of I/O lines

### DESCRIPTION

This I/O control function reads the value of the I/O lines. The function specific control parameter *arg* is a pointer on a *TDRV006_64BITBUFFER* structure.

```
typedef struct
{
        unsigned long       val_31_0;
        unsigned long       val_63_32;
} TDRV006_64BITBUFFER;
```

*val_31_0*

This argument returns the value of I/O lines 0 up to 31. Bit 0 returns the value of I/O line 0, bit 1 the value of I/O line 1, and so on.

*val_63_32*

This argument returns the value of I/O lines 32 up to 63. Bit 0 returns the value of I/O line 32, bit 1 the value of I/O line 33, and so on.

### EXAMPLE

```
#include “tdrv006.h”

int                  fd;
int                  result;
TDRV006_64BITBUFFER  rBuf;

/*--------------
  read I/O value
  --------------*/
result = ioctl(fd, TDRV006_READ, (char*)&rBuf);

if (result >= 0) {
     /* function succeeded */
     printf(“value: %08lX %08lXh\n”, rBuf.val_63_32, rBuf.val_31_0);
} else {
     /* handle the error */
}
```

## 3.3.2 TDRV006_WRITE

### NAME

TDRV006_WRITE – Write value of output lines

### DESCRIPTION

This I/O control function sets the value of the output lines. The function specific control parameter *arg* is a pointer on a *TDRV006_64BITBUFFER* structure.

```
typedef struct
{
        unsigned long       val_31_0;
        unsigned long       val_63_32;
} TDRV006_64BITBUFFER;
```

*val_31_0*

This argument specifies the output value of I/O lines 0 up to 31. Bit 0 specifies the value of I/O line 0, bit 1 the value of I/O line 1, and so on.

*val_63_32*

This argument specifies the output value of I/O lines 32 up to 63. Bit 0 specifies the value of I/O line 32, bit 1 the value of I/O line 33, and so on.

### EXAMPLE

```
#include “tdrv006.h”

int                    fd;
int                    result;
TDRV006_64BITBUFFER  wBuf;

/*--------------------------
  set I/O line 0-7 and 56-61
  --------------------------*/
wBuf.val_31_0   = 0x000000FF;
wBuf.val_63_32  = 0x3F000000;
result = ioctl(fd, TDRV006_WRITE, (char*)&wBuf);

if (result < 0) {
     /* handle the error */
}
```

## 3.3.3 TDRV006_OUTPUT_ENABLE

### NAME

TDRV006_OUTPUT_ENABLE – Enable output

### DESCRIPTION

This I/O control function configures the direction of the I/O lines. The function specific control parameter *arg* is a pointer on a *TDRV006_64BITBUFFER* structure.

```
typedef struct
{
        unsigned long       val_31_0;
        unsigned long       val_63_32;
} TDRV006_64BITBUFFER;
```

*val_31_0*

This argument specifies the direction of I/O lines 0 up to 31. Bit 0 specifies the direction of I/O line 0, bit 1 the direction of I/O line 1, and so on. A set bit configures the line for output, an unset bit configures input (tri-state).

*val_63_32*

This argument specifies the direction of I/O lines 32 up to 63. Bit 0 specifies the direction of I/O line 32, bit 1 the direction of I/O line 33, and so on. A set bit configures the line for output, an unset bit configures input (tri-state).

### EXAMPLE

```
#include “tdrv006.h”

int                   fd;
int                   result;
TDRV006_64BITBUFFER  dBuf;

/*------------------------------------------------
  configure line 0-12 for output, other are input
  -----------------------------------------------*/
dBuf.val_31_0    = 0x00001FFF;
dBuf.val_63_32   = 0x00000000;
result = ioctl(fd, TDRV006_OUTPUT_ENABLE, (char*)&dBuf);

if (result < 0) {
     /* handle the error */
}
```

## 3.3.4 TDRV006_EVENT_WAIT

### NAME

TDRV006_EVENT_WAIT – Wait for I/O transition events

### DESCRIPTION

This I/O control function waits for an I/O line transition event. The function specific control parameter *arg* is a pointer on a *TDRV006_EVENTWAITBUFFER* structure.

```
typedef struct
{
        int             mode;
        int             inputLine;
        long            timeout;
} TDRV006_EVENTWAITBUFFER;
```

*mode*

This argument specifies the transition the function should wait for. The following values are valid:

| Definition | Event |
|---|---|
| TDRV006_HIGH_TR | The function will return after a low to high transition has been detected |
| TDRV006_LOW_TR | The function will return after a high to low transition has been detected |
| TDRV006_ANY_TR | The function will return after a transition has been detected |

*inputLine*

This argument specifies the input line the event shall occur. Valid values are 0 up to 63.

*timeout*

This argument specifies the maximum time the function shall wait for the event. After this specified time the function will return with an error. The timeout time is specified in ticks. Specify -1 to wait indefinitely.

## EXAMPLE

```
#include “tdrv006.h”

int                         fd;
int                         result;
TDRV006_EVENTWAITBUFFER   evBuf;

/*------------------------------------
  Wait for any transition on I/O line 4
  ------------------------------------*/
evBuf.mode      = TDRV006_ANY_TR;
evBuf.inputLine = 4;
evBuf.timeout   = 600;      /* Ticks */

result = ioctl(fd, TDRV006_EVENT_WAIT, (char*)&evBuf);

if (result >= 0) {
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

## ERRORS

| | |
|---|---|
| EINTR | The function was cancelled. |
| EBUSY | Another task is already waiting for a transition of the specified input line |
| EINVAL | Invalid parameter specified |
| ETIMEDOUT | The maximum allowed time to finish the request is exhausted. |

Other returned error codes are system error conditions.

# 4 Debugging and Diagnostic

If the driver does not work properly, please enable debug outputs by defining the symbols *DEBUG, DEBUG_TPMC*, *DEBUG_PCI* and *DEBUG_INT.*

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, and a memory dump of the PCI base address registers like this.

```
TDRV006 Device Driver Install
Bus = 0  Dev = 11  Func = 0
[00] = 02A91498
[04] = 02800003
[08] = 11800000
[0C] = 00000008
[10] = EB021000
[14] = 0000B401
...
PCI Base Address 0 (PCI_RESID_BAR0)
E7021000 : 00 FF FF 0F 00 00 00 00 00 00 00 00 00 00 00 00
E7021010 : 00 00 00 00 01 00 00 00 00 00 00 00 00 00 00 00
E7021020 : 00 00 00 00 00 00 00 00 A0 20 81 15 00 00 00 00
E7021030 : 00 00 00 00 00 00 00 00 00 00 00 00 81 00 00 00
...
PCI Base Address 1 (PCI_RESID_BAR1)
0000B400 : 08 08 08 08 08 08 08 08 08 08 08 08 08 08 08 08
0000B410 : 08 08 08 08 08 08 08 08 08 08 08 08 08 08 08 08
0000B420 : 00 01 02 02 03 03 03 03 04 04 04 04 04 04 04 04
0000B430 : 05 05 05 05 05 05 05 05 05 05 05 05 05 05 05 05
...
PCI Base Address 2 (PCI_RESID_BAR2)
E7022000 : 00 00 00 00 00 00 00 00 FF FF FF FE FF FF FF FF
E7022010 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
E7022020 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
E7022030 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
...
Found a TPMC681, BusNo=0, DevNo=11
```

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