The Embedded I/O Company

TEWS TECHNOLOGIES

TDRV006-SW-82

Linux Device Driver

64 Digital Inputs/Outputs (Bit I/O)

Version 1.0.x

User Manual

Issue 1.0.3

January 2010

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

TDRV006-SW-82

Linux Device Driver

64 Digital Inputs/Outputs (Bit I/O)

Supported Modules:
TPMC681

This document contains information, which is proprietary to TEWS TECHNOLOGIES GmbH. Any reproduction without written permission is forbidden.

TEWS TECHNOLOGIES GmbH has made any effort to ensure that this manual is accurate and complete. However TEWS TECHNOLOGIES GmbH reserves the right to change the product described in this document at any time without notice.

TEWS TECHNOLOGIES GmbH is not liable for any damage arising out of the application or use of the device described herein.

Issue	Description	Date
1.0.0	First Issue	December 20, 2005
1.0.1	File list changed	August 15, 2006
1.0.2	New Address TEWS LLC	January 17, 2007
1.0.3	Address TEWS LLC removed	June 24, 2010

Table of Content

1 Introduction

The TDRV006-SW-82 Linux device driver allows the operation of the TDRV006 compatible PMCs conforming to the Linux I/O system specification. This includes a device-independent basic I/O interface with *open()*, *close()* and *ioctl()* functions.

Special I/O operation that do not fit to the standard I/O calls will be performed by calling the *ioctl()* function with a specific function code and an optional function dependent argument.

The TDRV006-SW-82 device driver supports the following features:

- Reading from input buffers
- Writing to output buffers
- Configuring I/O line directions
- Waiting for several input event types (PATTERN MATCH, RISING EDGE, FALLING EDGE)

The TDRV006-SW-82 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 the supported 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-82':

TDRV006-SW-82-1.0.3.pdf	This manual in PDF format
TDRV006-SW-82-SRC.tar.gz	GZIP compressed archive with driver source code
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 TDRV006-SW-82-SRC.tar.gz contains the following files and directories:

Directory path './tdrv006/':

tdrv006.c	Driver source code
tdrv006def.h	Driver include file
tdrv006.h	Driver include file for application program
makenode	Script to create device nodes on the file system
Makefile	Device driver make file
example/tdrv006exa.c	Example application
example/Makefile	Example application make file
include/tpmodule.h	Driver and kernel independent library header file
include/tpmodule.c	Driver and kernel independent library source file
include/tpxxxhwdep.h	HAL library header file
include/tpxxxhwdep.c	HAL library source file

In order to perform an installation, extract all files of the archive TDRV006-SW-82-SRC.tar.gz to the desired target directory.

- Login as *root* and change to the target directory
- Copy tdrv006.h to */usr/include*

2.1 Build and install the device driver

- Login as *root*
- Change to the target directory
- To create and install the driver in the module directory */lib/modules/<version>/misc* enter:

  # make install

- To update the device driver's module dependencies, enter:

  # **depmod -aq**

2.2 Uninstall the device driver

- Login as *root*
- Change to the target directory
- To remove the driver from the module directory */lib/modules/<version>/misc* enter:

```
# make uninstall
```

2.3 Install device driver into the running kernel

- To load the device driver into the running kernel, login as root and execute the following commands:

```
# modprobe tdrv006drv
```

- After the first build or if you are using dynamic major device allocation it is necessary to create new device nodes on the file system. Please execute the script file *makenode* to do this. If your kernel has enabled a device file system (devfs or sysfs with udev) then you have to skip running the *makenode* script. Instead of creating device nodes from the script the driver itself takes creating and destroying of device nodes in its responsibility.

```
# sh makenode
```

On success the device driver will create a minor device for each compatible channel found. The first channel of the first PMC module can be accessed with device node /dev/tdrv006_0, the second channel with device node /dev/tdrv006_1 and so on. The assignment of device nodes to physical PMC modules depends on the search order of the PCI bus driver.

2.4 Remove device driver from the running kernel

- To remove the device driver from the running kernel login as root and execute the following command:

```
# modprobe -r tdrv006drv
```

If your kernel has enabled devfs or sysfs (udev), all /dev/tdrv006_* nodes will be automatically removed from your file system after this.

Be sure that the driver is not opened by any application program. If opened you will get the response ``t*drv006drv: Device or resource busy*`` and the driver will still remain in the system until you close all opened files and execute *modprobe -r* again.

2.5 Change Major Device Number

This paragraph is only for Linux kernels without DEVFS installed.

The TPCM150 driver uses dynamic allocation of major device numbers per default. If this isn’t suitable for the application it’s possible to define a major number for the driver.

To change the major number edit the file tdrv006def.h, change the following symbol to appropriate value and enter `make install` to create a new driver.

TDRV006_MAJOR	Valid numbers are in range between 0 and 255. A value of 0 means dynamic number allocation.

Example:

```
#define TDRV006_MAJOR   122
```

Be sure that the desired major number is not used by other drivers. Please check *`/proc/devices`* to see which numbers are free.

Keep in mind that is necessary to create new device nodes if the major number for the TDRV006 driver has changed and the makenode script is not used.

2.6 Configuration

To adjust application specific driver properties see tdrv006def.h and look for the following symbol defines (#define <symbol> <value>):

TDRV006_MAX_EVENT_RECORDS

This symbol specifies the size of the interrupt routine event record queue. If you have input event loss during multiple TDRV006_IOC_EVENTWAIT jobs, please double the certain value.

TDRV006_MAX_EVENTWAIT_JOBS

This symbol specifies the maximum number of concurrent waiting threads in TDRV006_IOC_EVENTWAIT.

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 <fcntl.h>

int open
(
        const char *filename,
        int flags
)
```

DESCRIPTION

The open function creates and returns a new file descriptor for the file named by *filename*. The *flags* argument controls how the file is to be opened. This is a bit mask; you create the value by the bitwise OR of the appropriate parameters (using the **|** operator in C).

See also the GNU C Library documentation for more information about the open function and open flags.

EXAMPLE

```
int fd;


fd = open(“/dev/tdrv006_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

E_NODEV	The requested minor device does not exist.

This is the only error code returned by the driver, other codes may be returned by the I/O system during open. For more information about open error codes, see the *GNU C Library description – Low-Level Input/Output*.

SEE ALSO

GNU C Library description – Low-Level Input/Output

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

E_NODEV	The requested minor device does not exist.

This is the only error code returned by the driver, other codes may be returned by the I/O system during close. For more information about close error codes, see the *GNU C Library description – Low-Level Input/Output*.

SEE ALSO

GNU C Library description – Low-Level Input/Output

3.3 ioctl()

NAME

ioctl() – device control functions

SYNOPSIS

```
#include <sys/ioctl.h>

int ioctl
(
        int filedes,
        int request
        [, void *argp]
)
```

DESCRIPTION

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

The argument *request* specifies the control code for the operation. The optional argument *argp* depends on the selected request and is described for each request in detail later in this chapter.

The following ioctl codes are defined in *tdrv006.h* :

Symbol	**Meaning**
TDRV006_IOC_READ	Read value from input buffer
TDRV006_IOC_WRITE	Write value to output buffer
TDRV006_IOC_OE	Set pin direction
TDRV006_IOC_EVENTWAIT	Wait for input event

See behind for more detailed information on each control code.

To use these TDRV006 specific control codes the header file tdrv006.h must be included in the application.

RETURNS

On success, zero is returned. In the case of an error, a value of –1 is returned. The global variable *errno* contains the detailed error code.

ERRORS

EINVAL	Invalid argument. This error code is returned if the requested ioctl function is unknown. Please check the argument *request*
EFAULT	Parameter data can not be copied to the drivers context

Other function dependent error codes will be described for each ioctl code separately. Note, the TDRV006 driver always returns standard Linux error codes.

SEE ALSO

ioctl man pages

3.3.1 TDRV006_IOC_READ

NAME

TDRV006_IOC_ READ – Read value from input buffer

DESCRIPTION

This function reads the module input buffer. A pointer to the read buffer (TDRV006_UINT64) is passed by the parameter *arg* to the driver.

The TDRV006_UINT64 data type represents a 64-bit unsigned value. Each bit of the return value corresponds to an I/O line. Bit 0 represents the state of I/O line 0, Bit 1 belongs to I/O line 1 and so on.

EXAMPLE

```
#include <tdrv006.h>

int                     fd;
int                     result;
TDRV006_UINT64          ioBuffer;

printf("Read from input buffer ... ");

result = ioctl( fd, TDRV006_IOC_READ, &ioBuffer);
if (result >= 0)
{
     printf("OK\n");
     printf("  input value: "%08X%08X", (unsigned int)(ioBuffer >> 32),
                                        (unsigned int)ioBuffer);
}
else
{
     /* process ioctl error */
}
```

SEE ALSO

ioctl man pages

3.3.2 TDRV006_IOC_WRITE

NAME

TDRV006_IOC_ WRITE – Write value to output buffer

DESCRIPTION

This function writes to the output buffer. A pointer to the write buffer (TDRV006_UINT64) is passed by the parameter *arg* to the driver. Before writing to the output lines ensure the setting of the certain pin directions. You can set the pin direction (input or output) through TDRV006_OE ioctl function.

The TDRV006_UINT64 data type represents a 64-bit unsigned value. Each bit of the parameter value corresponds to an I/O line. Bit 0 represents the state of I/O line 0, Bit 1 belongs to I/O line 1 and so on.

EXAMPLE

```
#include <tdrv006.h>

int                     fd;
int                     result;
TDRV006_UINT64          ioBuffer;

printf("Write to output buffer ... ");

/* Set I/O lines 63, 34 to 32 and 0 to HIGH, all others set to LOW */
ioBuffer = ((TDRV006_UINT64)0x80000007 << 32) | (TDRV006_UINT64)0x00000001;

result = ioctl(fd, TDRV006_IOC_WRITE, &ioBuffer);
if (result >= 0)
{
    printf("OK\n");
}
else
{
    /* process ioctl error */
}
```

SEE ALSO

ioctl man pages

3.3.3 TDRV006_IOC_OE

NAME

TDRV006_IOC_ OE – Set pin directions

DESCRIPTION

This function sets the direction of each I/O line. A pointer to the direction buffer (TDRV006_UINT64) is passed by the parameter *arg* to the driver.

The TDRV006_UINT64 data type represents a 64-bit unsigned value. Each bit of the parameter value corresponds to an I/O line. Bit 0 represents I/O line 0, Bit 1 belongs to I/O line 1 and so on. A set bit in the certain position of the direction buffer enables the certain pin output buffer, otherwise the I/O line direction is set to input.

EXAMPLE

```
#include <tdrv006.h>

int                     fd;
int                     result;
TDRV006_UINT64          ioBuffer;

printf("Set pin direction ... ");

/* Set I/O lines 63, 34 to 32 and 0 to output, all others set to input */
ioBuffer = ((TDRV006_UINT64)0x80000007 << 32) | (TDRV006_UINT64)0x00000001;

result = ioctl(fd, TDRV006_IOC_OE, &ioBuffer);
if (result >= 0)
{
    printf("OK\n");
}
else
{
    /* process ioctl error */
}
```

SEE ALSO

ioctl man pages

3.3.4 TDRV006_IOC_EVENTWAIT

NAME

TDRV006_IOC_ EVENTWAIT – Wait for an input event

DESCRIPTION

This function waits a given amount of system ticks for a user defined input event. A pointer to the event buffer (TDRV006_EVENTWAIT) is passed by the parameter *arg* to the driver.

The TDRV006_UINT64 data type represents a 64-bit unsigned value. Each bit of a value corresponds to an I/O line. Bit 0 represents I/O line 0, Bit 1 belongs to I/O line 1 and so on. Ensure all bits used for input event detection are set to input through TDRV006_IOC_OE ioctl function.

```
struct {
    int                 mode;
    TDRV006_UINT64      mask;
    TDRV006_UINT64      code;
    TDRV006_UINT64      input;
    unsigned long       timeout;
} TDRV006_EVENTWAIT;
```

mode

This parameter specifies the event mode for this request.

TDRV006_RISING_EDGE	In this mode the ioctl function waits until a rising edge at one of the selected input line(s) or a timeout occurs.
TDRV006_FALLING_EDGE	In this mode the ioctl function waits until a falling edge at one of the selected input line(s) or a timeout occurs.
TDRV006_ANY_EDGE	In this mode the ioctl function waits until a falling or rising edge occurs at one of the selected input line(s) or a timeout occurs.
TDRV006_MATCH	In this mode the ioctl function waits until the masked bit group matches to the corresponding I/O-line group or a timeout occurs.

mask

This parameter specifies a bit mask to select a certain bit position or a group of bits for an input transition or match detection. A certain input line can be selected by setting the corresponding bit to 1, all others are don’t care bit positions.

code

This parameter specifies a bit code for input match detection. Don’t care bit position are masked by parameter *mask*. To achieve a match condition the following expression has to become TRUE.

((code & mask) == (<*input buffer state*> & mask))

This parameter in only used for TDRV006_MATCH mode.

input

After a successful completion of this request this parameter holds the state of the input buffer. Please note that the input buffer state isn’t latched with the interrupt and depending on the interrupt latency the read to the input buffer is delayed.

timeout

This parameter specifies the amount of time (in ticks) the caller is willing to wait for the occurrence of the requested transition or value match. A value of 0 means wait indefinitely.

EXAMPLE

```
#include <tdrv006.h>

int                     fd;
int                     result;
TDRV006_UINT64          ioBuffer;
TDRV006_EVENTWAIT       eW;

printf("Wait for input match event ... ");

/*
** Waiting for input match with code = 0x0200000230007000
** -lines 63 to 56 are don’t care -> mask = 0xF000000000000000
** -wait at least 1000 system ticks
*/
eW.mode = TDRV006_MATCH;
eW.mask = ((TDRV006_UINT64)0xF0000000 << 32) | (TDRV006_UINT64)0x00000000;
eW.code = ((TDRV006_UINT64)0x02000002 << 32) | (TDRV006_UINT64)0x30007000;
ew.timeout = 1000;

result = ioctl(fd, TDRV006_IOC_EVENTWAIT, &eW);
```

< *example continued on the next page* >

< continued >

```
if (result >= 0)
{
     printf("OK\n");
     printf("  input value: "%08X%08X", (unsigned int)(eW.input >> 32),
                                         (unsigned int)eW.input);
}
else
{
     /* process ioctl error */
}

printf("Wait for input transition event (RISING EDGE) ... ");

#define _BV64(n) ((TDRV006_UINT64)1 << n)
/*
** Waiting for rising edge on line 2,3,5,7,11,43
** -wait at least 1000 system ticks
*/
eW.mode = TDRV006_RISING_EDGE;
eW.mask = _BV64(2) | _BV64(3) | _BV64(5) | _BV64(7) | _BV64(11) |
_BV64(43);
ew.timeout = 1000;

result = ioctl(fd, TDRV006_IOC_EVENTWAIT, &eW);
if (result >= 0)
{
     printf("OK\n");
     printf("  input value: "%08X%08X", (unsigned int)(eW.input >> 32),
                                         (unsigned int)eW.input);
}
else
{
     /* process ioctl error */
}
```

ERROR

EINVAL
Invalid argument. This error code is returned if the requested ioctl function is unknown. Please check the argument *request.*

EFAULT
Parameter data can not be copied to or from the drivers context.

EAGAIN
Resource temporarily unavailable; the call might work if you try again later. This error occurs only if the device is opened with the flag O_NONBLOCK set.

ETIME
The allowed time to finish the input event request has elapsed.

EINTR
Interrupted function call; an asynchronous signal occurred and prevented completion of the call. When this happens, you should try the call again.

SEE ALSO

ioctl man pages

4 Diagnostic

If the TDRV006 does not work properly it is helpful to get some status information from the driver respective kernel.

The Linux */proc* file system provides information about kernel, resources, driver, devices and so on. The following screen dumps displays information of a correct running TDRV006 driver (see also the proc man pages).

```
# lspci -v
...
02:08.0 Signal processing controller: TEWS Datentechnik GmBH: Unknown
device 02a9
        Subsystem: TEWS Datentechnik GmBH: Unknown device 000a
        Flags: medium devsel, IRQ 177
        Memory at ff5fe400 (32-bit, non-prefetchable)
        I/O ports at a800 [size=128]
        Memory at ff5fe000 (32-bit, non-prefetchable) [size=256]...

02:09.0 Signal processing controller: TEWS Datentechnik GmBH: Unknown
device 02a9
        Subsystem: TEWS Datentechnik GmBH: Unknown device 000a
        Flags: medium devsel, IRQ 169
        Memory at ff5fec00 (32-bit, non-prefetchable)
        I/O ports at a880 [size=128]
        Memory at ff5fe800 (32-bit, non-prefetchable) [size=256]
...

# cat /proc/devices
Character devices:
  1 mem
  2 pty
  3 ttyp
  4
  5 cua
  7 vcs
 10 misc
 13 input
 14 sound
 29 fb
 36 netlink
162 raw
180 usb
226 drm
254 tdrv006drv
```

```
# cat /proc/interrupts
           CPU0       CPU1
  0:    4482728    4529560    IO-APIC-edge  timer
  1:          0         10    IO-APIC-edge  i8042
  2:          0          0          XT-PIC  cascade
  8:          0          1    IO-APIC-edge  rtc
  9:         70         58   IO-APIC-level  acpi
 12:          0         58    IO-APIC-edge  i8042
 14:       2708       8067    IO-APIC-edge  ide0
169:     577517     581029   IO-APIC-level  radeon@PCI:1:0:0, TDRV006
177:         85         43   IO-APIC-level  uhci_hcd, TDRV006
185:      11832         29   IO-APIC-level  uhci_hcd, eth0
193:          0          0   IO-APIC-level  libata, ehci_hcd, ohci_hcd,
ohci_hcd
NMI:          0          0
LOC:    9011342    9011340
ERR:          0
MIS:          0

# cat /proc/ioports
...
03f6-03f6 : ide0
03f8-03ff : serial
0cf8-0cff : PCI conf1
7000-9fff : PCI Bus #01
  9000-90ff : 0000:01:00.0
a000-bfff : PCI Bus #02
  a400-a43f : 0000:02:03.0
    a400-a43f : e1000
  a480-a4bf : 0000:02:06.0
    a480-a4bf : e100
  a800-a87f : 0000:02:08.0
  a880-a8ff : 0000:02:09.0
  ac00-ac7f : 0000:02:0a.0
  b000-b01f : 0000:02:0b.0
    b000-b01f : uhci_hcd
  b080-b09f : 0000:02:0b.1
    b080-b09f : uhci_hcd
  b400-b40f : 0000:02:05.0
    b400-b40f : sata_sil
  b480-b483 : 0000:02:05.0
  dc00-dcff : SiS 7012
...
```

```
# cat /proc/iomem

00000000-0009fbff : System RAM
0009fc00-0009ffff : reserved
000a0000-000bffff : Video RAM area
000c0000-000cbfff : Video ROM
000cc000-000cd7ff : Adapter ROM
000cd800-000ce7ff : Adapter ROM
000f0000-000fffff : System ROM
00100000-3ffeffff : System RAM
  00100000-002a2fff : Kernel code
  002a3000-003542ff : Kernel data
3fff0000-3fffefff : ACPI Tables
3ffff000-3fffffff : ACPI Non-volatile Storage
deb00000-eeafffff : PCI Bus #01
  e0000000-e7ffffff : 0000:01:00.0
f0000000-f7ffffff : 0000:00:00.0
  f0000000-f7ffffff : aperture
ff200000-ff2fffff : PCI Bus #01
  ff2f0000-ff2fffff : 0000:01:00.0
ff300000-ff5fffff : PCI Bus #02
  ff580000-ff59ffff : 0000:02:03.0
    ff580000-ff59ffff : e1000
  ff5a0000-ff5bffff : 0000:02:03.0
    ff5a0000-ff5bffff : e1000
  ff5c0000-ff5dffff : 0000:02:06.0
    ff5c0000-ff5dffff : e100
  ff5fb000-ff5fbfff : 0000:02:00.0
    ff5fb000-ff5fbfff : ohci_hcd
  ff5fc000-ff5fcfff : 0000:02:00.1
    ff5fc000-ff5fcfff : ohci_hcd
  ff5fd000-ff5fdfff : 0000:02:06.0
    ff5fd000-ff5fdfff : e100
  ff5fe000-ff5fe0ff : 0000:02:08.0
    ff5fe000-ff5fe0ff : TDRV006
  ff5fe400-ff5fe47f : 0000:02:08.0
  ff5fe800-ff5fe8ff : 0000:02:09.0
    ff5fe800-ff5fe8ff : TDRV006
  ff5fec00-ff5fec7f : 0000:02:09.0
...
```