*The Embedded I/O Company*

TEWS TECHNOLOGIES

# TDRV012-SW-82

## Linux Device Driver

32 differential I/O Lines with Interrupts

Version 2.0.x

## User Manual

Issue 2.0.1

August 2018

powerBridge Computer

Ehlbeek 15a
30938 Burgwedel
fon 05139-9980-0
fax 05139-9980-49

www.powerbridge.de
info@powerbridge.de

**TEWS TECHNOLOGIES GmbH**
Am Bahnhof 7 25469 Halstenbek, Germany
(0) 4101 4058 0 Fax: +49 (0) 4101 4058 19
info@tews.com www.tews.com

## TDRV012-SW-82

Linux Device Driver

32 differential I/O Lines with Interrupts

Supported Modules:

TPMC683



| Issue | Description | Date |
|---|---|---|
| 1.0.0 | First Issue | March 20, 2009 |
| 2.0.0 | API implemented, General Revision | December 18, 2012 |
| 2.0.1 | File-List modified | August 29, 2018 |

# Table of Contents


1 INTRODUCTION ..... 4

2 INSTALLATION ..... 5

2.1 Build and install the device driver ..... 5
2.2 Uninstall the device driver ..... 6
2.3 Install device driver into the running kernel ..... 6
2.4 Remove device driver from the running kernel ..... 6
2.5 Change Major Device Number ..... 7

3 API DOCUMENTATION ..... 8

3.1 General Functions ..... 8
3.1.1 tdrv012Open ..... 8
3.1.2 tdrv012Close ..... 10
3.2 Device Access Functions ..... 12
3.2.1 tdrv012Read ..... 12
3.2.2 tdrv012WriteMask ..... 14
3.2.3 tdrv012OutputSet ..... 16
3.2.4 tdrv012OutputClear ..... 18
3.2.5 tdrv012ConfigureDirection ..... 20
3.2.6 tdrv012ReadDirection ..... 22
3.2.7 tdrv012WaitEvent ..... 24
3.2.8 tdrv012WaitHigh ..... 27
3.2.9 tdrv012WaitLow ..... 29
3.2.10 tdrv012WaitAny ..... 31

4 DIAGNOSTIC ..... 33

# 1 Introduction

The TDRV012-SW-82 Linux device driver allows the operation of the TDRV012 compatible devices conforming to the Linux I/O system specification.

The TDRV012-SW-82 device driver supports the following features:

- configure input/output direction of each line
- read state of input lines
- write to output lines
- wait for interrupt events (rising/falling edge) on each input line

The TDRV012-SW-82 supports the modules listed below:

| | | |
|---|---|---|
| TPMC683 | 32 differential I/O Lines with Interrupts | PMC |

**In this document all supported modules and devices will be called TDRV012. Specials for a certain device will be advised.**

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

| |
|---|
| TPMC683 User Manual |

# 2 Installation

Following files are located on the distribution media:

Directory path 'TDRV012-SW-82':

| | |
|---|---|
| TDRV012-SW-82-2.0.1.pdf | This manual in PDF format |
| TDRV012-SW-82-SRC.tar.gz | GZIP compressed archive with driver source code |
| ChangeLog.txt | Release history |
| Release.txt | Information about the Device Driver Release |

The GZIP compressed archive TDRV012-SW-82-SRC.tar.gz contains the following files and directories:

Directory path 'tdrv012':

| | |
|---|---|
| tdrv012.c | Driver source code |
| tdrv012def.h | Driver include file |
| tdrv012.h | Driver include file for application program |
| Makefile | Device driver make file |
| makenode | Script for device node creation |
| include/config.h | Driver independent configuration header file |
| include/tpmodule.c | Driver independent library |
| include/tpmodule.h | Driver independent library header file |
| include/tpxxxhwdep.h | HAL library header file |
| include/tpxxxhwdep.c | HAL library source file |
| api/tdrv012api.h | API include file |
| api/tdrv012api.c | API source file |
| example/tdrv012exa.c | Example application |
| example/Makefile | Example application makefile |
| COPYING | Copy of the GNU Public License (GPL) |

In order to perform an installation, extract all files of the archive TDRV012-SW-82-SRC.tar.gz to the desired target directory. The command 'tar -xzvf TDRV012-SW-82-SRC.tar.gz' will extract the files into the local directory.

- Login as *root* and change to the target directory
- Copy tdrv012.h and api/tdrv012api.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 tdrv012drv**

- 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 dynamic 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 TDRV012 device found. The first TDRV012 device can be accessed with device node /dev/tdrv012_0, the second module with device node /dev/tdrv012_1 and so on.

The assignment of device nodes to physical TDRV012 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 tdrv012drv**

If your kernel has enabled a dynamic device file system (devfs or sysfs with udev) all /dev/tdrv012_x nodes will be automatically removed from your file system after this.

**Be sure that the driver isn't opened by any application program. If opened you will get the response "*tdrv012drv: 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 installed dynamic device file system. The TDRV012 driver uses dynamic allocation of major device numbers per default. If this isn't suitable for the application it is possible to define a major number for the driver.

To change the major number, edit the file tdrv012def.h, change the following symbol to appropriate value and enter `make install` to create a new driver.

| | |
|---|---|
| TDRV012_MAJOR | Valid numbers are in range between 0 and 255. A value of 0 means dynamic number allocation. |

**EXAMPLE:**

```
#define TDRV012_MAJOR   122
```

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

# 3 API Documentation

## 3.1 General Functions

### 3.1.1 tdrv012Open

#### NAME

tdrv012Open – Open a Device

#### SYNOPSIS

```
TDRV012_HANDLE tdrv012Open
(
      char                          *DeviceName
)
```

#### DESCRIPTION

Before I/O can be performed to a device, a file descriptor must be opened by a call to this function.

#### PARAMETERS

*DeviceName*

This parameter points to a null-terminated string that specifies the name of the device. The first TDRV012 device is named /dev/tdrv012_0, the second /dev/tdrv012_1, and so on.

#### EXAMPLE

```
#include <tdrv012api.h>

TDRV012_HANDLE hdl;

/*
** open file descriptor to device
*/
hdl = tdrv012Open("/dev/tdrv012_0");
if (hdl == NULL)
{
  /* handle open error */
}
```

## RETURNS

A device handle, or NULL if the function fails. An error code will be stored in *errno*.

## ERROR CODES

All error codes are standard error codes set by the I/O system.

## 3.1.2 tdrv012Close

### NAME

tdrv012Close – Close a Device

### SYNOPSIS

```
TDRV012_STATUS tdrv012Close
(
      TDRV012_HANDLE              hdl
)
```

### DESCRIPTION

This function closes previously opened devices.

### PARAMETERS

*hdl*

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

### EXAMPLE

```
#include <tdrv012api.h>

TDRV012_HANDLE   hdl;
TDRV012_STATUS   result;

/*
** close file descriptor to device
*/
result = tdrv012Close( hdl );
if (result != TDRV012_OK)
{
   /* handle close error */
}
```

## RETURNS

On success TDRV012_OK, or an appropriate error code.

## ERROR CODES

| Error Code | Description |
|---|---|
| TDRV012_ERR_INVALID_HANDLE | The specified TDRV012_HANDLE is invalid. |

Other returned error codes are system error conditions.

# 3.2 Device Access Functions

## 3.2.1 tdrv012Read

### NAME

tdrv012Read – Read current I/O Value

### SYNOPSIS

```
TDRV012_STATUS tdrv012Read
(
    TDRV012_HANDLE      hdl,
    unsigned int        *ploValue
)
```

### DESCRIPTION

This function reads the current state of the input and output lines of the specified device.

### PARAMETERS

*hdl*

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

*ploValue*

This value is a pointer to a uint32_t 32bit data buffer which receives the current I/O value. Both input and output values are returned. Bit 0 of this value corresponds to the first I/O line, bit 1 corresponds to the second I/O line and so on.

## EXAMPLE

```
#include <tdrv012api.h>

TDRV012_HANDLE   hdl;
TDRV012_STATUS   result;
unsigned int     IoValue;

/*
** read current I/O value
*/
result = tdrv012Read( hdl, &IoValue );
if (result == TDRV012_OK)
{
  printf( "I/O Value: 0x%08X\n", IoValue );
} else {
  /* handle error */
}
```

## RETURNS

On success, TDRV012_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

| Error Code | Description |
|---|---|
| TDRV012_ERR_INVALID_HANDLE | The specified TDRV012_HANDLE is invalid. |

Other returned error codes are system error conditions.

## 3.2.2 tdrv012WriteMask

### NAME

tdrv012WriteMask – Write relevant Bits of Output Value

### SYNOPSIS

```
TDRV012_STATUS tdrv012WriteMask
(
        TDRV012_HANDLE          hdl,
        unsigned int            OutputValue,
        unsigned int            BitMask
);
```

### DESCRIPTION

This function writes relevant bits of a new output value for the specified device.

### PARAMETERS

*hdl*

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

*OutputValue*

This value specifies the new output value. Bit 0 of this value corresponds to the first output line, bit 1 corresponds to the second output line and so on.

*BitMask*

This parameter specifies the bitmask. Only active bits (1) will be written to the output register, all other output lines will be left unchanged. Bit 0 of this value corresponds to the first output line, bit 1 corresponds to the second output line and so on.

## EXAMPLE

```
#include <tdrv012api.h>

TDRV012_HANDLE  hdl;
TDRV012_STATUS  result;

/*
** write new output value:
** set 2nd  (bit 1) output line to ON, and 7th (bit 6) output line to OFF.
** leave all other output lines unchanged.
*/
result = tdrv012WriteMask(
       hdl,
       (1 << 1),
       (1 << 1) | (1 << 6)
    );
if (result == TDRV012_OK)
{
  /* OK */
} else {
  /* handle error */
}
```

## RETURNS

On success, TDRV012_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

| Error Code | Description |
|---|---|
| TDRV012_ERR_INVALID_HANDLE | The specified TDRV012_HANDLE is invalid. |

Other returned error codes are system error conditions.

## 3.2.3 tdrv012OutputSet

### NAME

tdrv012OutputSet – Set single Output Lines to ON

### SYNOPSIS

```
TDRV012_STATUS tdrv012OutputSet
(
    TDRV012_HANDLE      hdl,
    unsigned int        OutputValue
)
```

### DESCRIPTION

This function sets single output lines to ON leaving other output lines in the current state.

### PARAMETERS

*hdl*

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

*OutputValue*

This value specifies the new output value. Active (1) bits will set the corresponding output line to ON, unset (0) bits will not have an effect on the corresponding output lines. Bit 0 of this value corresponds to the first output line, bit 1 corresponds to the second output line and so on.

## EXAMPLE

```
#include <tdrv012api.h>

TDRV012_HANDLE  hdl;
TDRV012_STATUS  result;

/*
** write new output value:
** set 2nd  (bit 1) and 3rd (bit 2) output line to ON.
** leave all other output lines unchanged.
*/
result = tdrv012OutputSet(
       hdl,
       (1 << 1) | (1 << 2)
    );
if (result == TDRV012_OK)
{
  /* OK */
} else {
  /* handle error */
}
```

## RETURNS

On success, TDRV012_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

| Error Code | Description |
|---|---|
| TDRV012_ERR_INVALID_HANDLE | The specified TDRV012_HANDLE is invalid. |

Other returned error codes are system error conditions.

## 3.2.4 tdrv012OutputClear

### NAME

tdrv012OutputClear – Clear single Output Lines to OFF

### SYNOPSIS

```
TDRV012_STATUS tdrv012OutputClear
(
    TDRV012_HANDLE      hdl,
    unsigned int        OutputValue
)
```

### DESCRIPTION

This function clears single output lines to OFF leaving other output lines in the current state.

### PARAMETERS

*hdl*

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

*OutputValue*

This value specifies the new output value. Active (1) bits will clear the corresponding output line to OFF, unset (0) bits will not have an effect on the corresponding output lines. Bit 0 of this value corresponds to the first output line, bit 1 corresponds to the second output line and so on.

## EXAMPLE

```
#include <tdrv012api.h>

TDRV012_HANDLE  hdl;
TDRV012_STATUS  result;

/*
** write new output value:
** clear 2nd  (bit 1) and 4th (bit 3) output line to OFF.
** leave all other output lines unchanged.
*/
result = tdrv012OutputClear(
       hdl,
       (1 << 1) | (1 << 3)
    );
if (result == TDRV012_OK)
{
  /* OK */
} else {
  /* handle error */
}
```

## RETURNS

On success, TDRV012_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

| Error Code | Description |
|---|---|
| TDRV012_ERR_INVALID_HANDLE | The specified TDRV012_HANDLE is invalid. |

Other returned error codes are system error conditions.

## 3.2.5 tdrv012ConfigureDirection

### NAME

tdrv012ConfigureDirection – Configure Input/Output Direction of I/O Lines

### SYNOPSIS

```
TDRV012_STATUS tdrv012ConfigureDirection
(
    TDRV012_HANDLE      hdl,
    unsigned int        DirectionValue,
    unsigned int        DirectionMask
)
```

### DESCRIPTION

This function configures the direction (input/output) of specific I/O lines. Only specific lines specified by a mask are affected.

### PARAMETERS

*hdl*

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

*DirectionValue*

This value specifies the direction of the corresponding I/O lines. An active (1) bit will configure the corresponding I/O line to OUTPUT, an unset (0) bit will configure the corresponding I/O line to INPUT. Bit 0 of this value corresponds to the first I/O line, bit 1 corresponds to the second I/O line and so on.

*DirectionMask*

This parameter specifies the bitmask. Only active bits (1) will have an effect on the I/O direction, the direction of all other I/O lines will be left unchanged. Bit 0 of this value corresponds to the first I/O line, bit 1 corresponds to the second I/O line and so on.

## EXAMPLE

```
#include <tdrv012api.h>

TDRV012_HANDLE  hdl;
TDRV012_STATUS  result;

/*
** configure new I/O direction:
** set lowest 8 I/O lines to OUTPUT, and highest 8 I/O lines to INPUT.
** leave all other I/O lines unchanged.
*/
result = tdrv012ConfigureDirection(
       hdl,
       (0x00 << 24) | (0xff << 0),
       (0xff << 24) | (0xff << 0)
    );
if (result == TDRV012_OK)
{
  /* OK */
} else {
  /* handle error */
}
```

## RETURNS

On success, TDRV012_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

| Error Code | Description |
|---|---|
| TDRV012_ERR_INVALID_HANDLE | The specified TDRV012_HANDLE is invalid. |

Other returned error codes are system error conditions.

## 3.2.6 tdrv012ReadDirection

### NAME

tdrv012ReadDirection – Read current Input/Output Direction Configuration of I/O Lines

### SYNOPSIS

```
TDRV012_STATUS tdrv012ReadDirection
(
    TDRV012_HANDLE      hdl,
    unsigned int        *pDirectionValue
)
```

### DESCRIPTION

This function reads the current direction configuration (input/output) of the I/O lines.

### PARAMETERS

*hdl*

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

*pDirectionValue*

This value is a pointer to an *unsigned int* 32bit data buffer which receives the current I/O direction configuration. Active (1) bits represent OUTPUT lines, unset (0) bits represent INPUT lines. Bit 0 of this value corresponds to the first I/O line, bit 1 corresponds to the second I/O line and so on.

## EXAMPLE

```
#include <tdrv012api.h>

TDRV012_HANDLE   hdl;
TDRV012_STATUS   result;
unsigned int     DirectionValue;

/*
** read current I/O direction configuration
*/
result = tdrv012ReadDirection(
       hdl,
       &DirectionValue
     );
if (result == TDRV012_OK)
{
  printf("Current direction configuration (1=OUTPUT, 0=INPUT):\n");
  printf("  0x%08X\n", DirectionValue);
} else {
  /* handle error */
}
```

## RETURNS

On success, TDRV012_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

| Error Code | Description |
|---|---|
| TDRV012_ERR_INVALID_HANDLE | The specified TDRV012_HANDLE is invalid. |

Other returned error codes are system error conditions.

## 3.2.7 tdrv012WaitEvent

### NAME

tdrv012WaitEvent – Wait for specific Transitions on I/O Lines

### SYNOPSIS

```
TDRV012_STATUS tdrv012WaitEvent
(
    TDRV012_HANDLE      hdl,
    unsigned int        mask_high,
    unsigned int        mask_low,
    int                 timeout,
    unsigned int        *pIoValue,
    unsigned int        *pStatusHigh,
    unsigned int        *pStatusLow
);
```

### DESCRIPTION

This function blocks until at least one of the specified events or a timeout occurs.

### PARAMETERS

*hdl*

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

*mask_high*

This parameter specifies on which input line a HIGH transition should occur to trigger an event. Multiple input lines may be specified. Bit 0 of this value corresponds to the first I/O line, bit 1 corresponds to the second I/O line and so on.

*mask_low*

This parameter specifies on which input line a LOW transition should occur to trigger an event. Multiple input lines may be specified. Bit 0 of this value corresponds to the first I/O line, bit 1 corresponds to the second I/O line and so on.

*timeout*

This parameter specifies the time the function should wait for the event. The timeout is specified in milliseconds. Use -1 to wait indefinitely for the event.

*pIoValue*

This value is a pointer to an *unsigned int* 32bit data buffer which returns the state of the input lines at the moment the event is served by the interrupt service routine. Keep in mind that there is a system-dependent interrupt latency, so it is not guaranteed that this value is the actual input state at the time of the event.

*pStatusHigh*

This parameter is a pointer to an *unsigned int* 32bit data buffer which returns on which input lines a HIGH transition has occurred for the current wait job. This parameter is a bitmask, where bit 0 corresponds to I/O line 0, bit 1 corresponds to I/O line 1 and so on.

*pStatusLow*

This parameter is a pointer to an *unsigned int* 32bit data buffer which returns on which input lines a LOW transition has occurred for the current wait job. This parameter is a bitmask, where bit 0 corresponds to I/O line 0, bit 1 corresponds to I/O line 1 and so on.

## EXAMPLE

```
#include <tdrv012api.h>

TDRV012_HANDLE   hdl;
TDRV012_STATUS   result;
unsigned int     IoValue, StatusHigh, StatusLow;

/*
** wait at least 1000ms for events:
** HIGH transition on I/O line 0 or
** LOW transition on I/O line 1 or
** HIGH/LOW=ANY transition on I/O line 2
*/
result = tdrv012WaitEvent(
        hdl,
        (1 << 2) | (1 << 0),
        (1 << 2) | (1 << 1),
        1000,
        &IoValue,
        &StatusHigh,
        &StatusLow
     );

...
```

…

```
if (result == TDRV012_OK)
{
   printf("  Current I/O status    : 0x%08X\n", IoValue);
   printf("  HIGH transition status: 0x%08X\n", StatusHigh);
   printf("  LOW  transition status: 0x%08X\n", StatusLow);
} else {
   /* handle error */
}
```

## RETURNS

On success, TDRV012_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

| Error Code | Description |
|---|---|
| TDRV012_ERR_INVALID_HANDLE | The specified TDRV012_HANDLE is invalid. |
| TDRV012_ERR_BUSY | Too many concurrent wait jobs pending (max. 100) |
| TDRV012_ERR_TIMEOUT | Timeout. None of the specified events occurred. |

Other returned error codes are system error conditions.

## 3.2.8 tdrv012WaitHigh

### NAME

tdrv012WaitHigh – Wait for HIGH Transitions on specific I/O Lines

### SYNOPSIS

```
TDRV012_STATUS tdrv012WaitHigh
(
    TDRV012_HANDLE      hdl,
    unsigned int        mask,
    int                 timeout,
    unsigned int        *pIoValue,
    unsigned int        *pStatus
);
```

### DESCRIPTION

This function blocks until at least one of the specified HIGH events or a timeout occurs.

### PARAMETERS

*hdl*

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

*mask*

This parameter specifies on which input line the HIGH transition should occur to trigger an event. Multiple input lines may be specified. Bit 0 of this value corresponds to the first I/O line, bit 1 corresponds to the second I/O line and so on.

*timeout*

This parameter specifies the time the function should wait for the event. The timeout is specified in milliseconds. Use -1 to wait indefinitely for the event.

*pIoValue*

This value is a pointer to an *unsigned int* 32bit data buffer which returns the state of the input lines at the moment the event is served by the interrupt service routine. Keep in mind that there is a system-dependent interrupt latency, so it is not guaranteed that this value is the actual input state at the event.

*pStatus*

This parameter is a pointer to an *unsigned int* 32bit data buffer which returns on which input lines a HIGH transition has occurred for the current wait job. This parameter is a bitmask, where bit 0 corresponds to I/O line 0, bit 1 corresponds to I/O line 1 and so on.

## EXAMPLE

```
#include <tdrv012api.h>

TDRV012_HANDLE  hdl;
TDRV012_STATUS  result;
unsigned int    IoValue;
unsigned int    Status;

/*
** wait at least 1000ms for HIGH transition events:
** HIGH transition on I/O line 31
*/
result = tdrv012WaitHigh(
       hdl,
       (1 << 31),
       1000,
       &IoValue,
       &Status
    );
if (result == TDRV012_OK)
{
  printf("  Current I/O status    : 0x%08X\n", IoValue);
  printf("  HIGH transition status: 0x%08X\n", Status);
} else {
  /* handle error */
}
```

## RETURNS

On success, TDRV012_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

| Error Code | Description |
|---|---|
| TDRV012_ERR_INVALID_HANDLE | The specified TDRV012_HANDLE is invalid. |
| TDRV012_ERR_BUSY | Too many concurrent wait jobs pending (max. 100) |
| TDRV012_ERR_TIMEOUT | Timeout. None of the specified events occurred. |

Other returned error codes are system error conditions.

## 3.2.9 tdrv012WaitLow

### NAME

tdrv012WaitLow – Wait for LOW Transitions on specific I/O Lines

### SYNOPSIS

```
TDRV012_STATUS tdrv012WaitLow
(
    TDRV012_HANDLE      hdl,
    unsigned int        mask,
    int                 timeout,
    unsigned int        *pIoValue,
    unsigned int        *pStatus
)
```

### DESCRIPTION

This function blocks until at least one of the specified LOW events or a timeout occurs.

### PARAMETERS

*handle*

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

*mask*

This parameter specifies on which input line the LOW transition should occur to trigger an event. Multiple input lines may be specified. Bit 0 of this value corresponds to the first I/O line, bit 1 corresponds to the second I/O line and so on.

*timeout*

This parameter specifies the time the function should wait for the event. The timeout is specified in milliseconds. Use -1 to wait indefinitely for the event.

*pIoValue*

This value is a pointer to an *unsigned int* 32bit data buffer which returns the state of the input lines at the moment the event is served by the interrupt service routine. Keep in mind that there is a system-dependent interrupt latency, so it is not guaranteed that this value is the actual input state at the event.

*pStatus*

This parameter is a pointer to an *unsigned int* 32bit data buffer which returns on which input lines a LOW transition has occurred for the current wait job. This parameter is a bitmask, where bit 0 corresponds to I/O line 0, bit 1 corresponds to I/O line 1 and so on.

## EXAMPLE

```
#include <tdrv012api.h>

TDRV012_HANDLE  hdl;
TDRV012_STATUS  result;
unsigned int    IoValue;
unsigned int    Status;

/*
** wait at least 1000ms for LOW transition events:
** LOW transition on I/O line 31
*/
result = tdrv012WaitLow(
       hdl,
       (1 << 31),
       1000,
       &IoValue,
       &Status
     );
if (result == TDRV012_OK)
{
   printf("  Current I/O status   : 0x%08X\n", IoValue);
   printf("  LOW transition status: 0x%08X\n", Status);
} else {
   /* handle error */
}
```

## RETURNS

On success, TDRV012_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

| Error Code | Description |
|---|---|
| TDRV012_ERR_INVALID_HANDLE | The specified TDRV012_HANDLE is invalid. |
| TDRV012_ERR_BUSY | Too many concurrent wait jobs pending (max. 100) |
| TDRV012_ERR_TIMEOUT | Timeout. None of the specified events occurred. |

Other returned error codes are system error conditions.

## 3.2.10 tdrv012WaitAny

### NAME

tdrv012WaitAny – Wait for HIGH or LOW Transitions on specific I/O Lines

### SYNOPSIS

```
TDRV012_STATUS tdrv012WaitAny
(
    TDRV012_HANDLE      hdl,
    unsigned int        mask,
    int                 timeout,
    unsigned int        *pIoValue,
    unsigned int        *pStatus
)
```

### DESCRIPTION

This function blocks until at least one of the specified HIGH or LOW events or a timeout occurs.

### PARAMETERS

*hdl*

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

*mask*

This parameter specifies on which input line the HIGH or LOW transition should occur to trigger an event. Multiple input lines may be specified. Bit 0 of this value corresponds to the first I/O line, bit 1 corresponds to the second I/O line and so on.

*timeout*

This parameter specifies the time the function should wait for the event. The timeout is specified in milliseconds, although the granularity is in seconds. Use -1 to wait indefinitely for the event.

*pIoValue*

This value is a pointer to an *unsigned int* 32bit data buffer which returns the state of the input lines at the moment the event is served by the interrupt service routine. Keep in mind that there is a system-dependent interrupt latency, so it is not guaranteed that this value is the actual input state at the event.

*pStatus*

This parameter is a pointer to an *unsigned int* 32bit data buffer which returns on which input lines a HIGH or LOW transition has occurred for the current wait job. This parameter is a bitmask, where bit 0 corresponds to I/O line 0, bit 1 corresponds to I/O line 1 and so on. It is not possible to distinguish between a HIGH or LOW event. To do this, use tdrv012waitEvent() instead.

## EXAMPLE

```
#include <tdrv012api.h>

TDRV012_HANDLE  hdl;
TDRV012_STATUS  result;
unsigned int    IoValue;
unsigned int    Status;

/*
** wait at least 1000ms for HIGH or LOW transition events:
** any transition on I/O line 0
*/
result = tdrv012WaitAny(
       hdl,
       (1 << 0),
       1000,
       &IoValue,
       &Status
    );
if (result == TDRV012_OK)
{
  printf("  Current I/O status   : 0x%08X\n", IoValue);
  printf("  transition status    : 0x%08X\n", Status);
} else {
  /* handle error */
}
```

## RETURNS

On success, TDRV012_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

| Error Code | Description |
|---|---|
| TDRV012_ERR_INVALID_HANDLE | The specified TDRV012_HANDLE is invalid. |
| TDRV012_ERR_BUSY | Too many concurrent wait jobs pending (max. 100) |
| TDRV012_ERR_TIMEOUT | Timeout. None of the specified events occurred. |

Other returned error codes are system error conditions.

# 4 Diagnostic

If the TDRV012 does not work properly it is helpful to get some status information from the driver respective kernel.

The Linux */proc* file system provides additional information about kernel, resources, drivers, devices and so on. The following screen dumps display information of a correct running TDRV012 driver (see also the proc man pages).

```
# lspci -v
...
04:02.0 Signal processing controller: TEWS Technologies GmbH Device 02ab
        Subsystem: TEWS Technologies GmbH Device 000a
        Flags: medium devsel, IRQ 17
        Memory at febefc00 (32-bit, non-prefetchable) [size=128]
        I/O ports at ec00 [size=128]
        Memory at febef800 (32-bit, non-prefetchable) [size=256]
        Kernel driver in use: TEWS TECHNOLOGIES - TDRV012 Device Driver
        Kernel modules: tdrv012drv
...

# cat /proc/devices
Character devices:
  1 mem
  2 pty
  . . .
136 pts
162 raw
  . . .
248 tdrv012drv


# cat /proc/iomem
00000000-0009fbff : System RAM
  . . .
feb00000-febfffff : PCI Bus 0000:04
  febff000-febff0ff : 0000:04:02.0
    febff000-febff0ff : TDRV012
. . .
```