

The Embedded I/O Company



TDRV012-SW-65

Windows Device Driver

32 differential I/O Lines with Interrupts

Version 2.0.x

User Manual

Issue 2.0.0

March 2011

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
9 (0) 4101 4058 0 Fax: +49 (0) 4101 4058 19
il: info@tews.com www.tews.com

TDRV012-SW-65

Windows Device Driver

32 differential I/O Lines with Interrupts

Supported Modules:
TPMC683

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.

©2009-2011 by TEWS TECHNOLOGIES GmbH

Issue	Description	Date
1.0.0	First Issue	September 25, 2009
2.0.0	Support for Windows 7 added, API functions and parameters modified	March 04, 2011

Table of Contents

1	INTRODUCTION.....	4
2	INSTALLATION.....	5
	2.1 Software Installation.....	5
	2.1.1 Windows 2000 / XP.....	5
	2.1.2 Windows 7.....	6
	2.2 Confirming Driver Installation.....	6
3	API DOCUMENTATION.....	7
	3.1 General Functions.....	7
	3.1.1 tdrv012Open.....	7
	3.1.2 tdrv012Close.....	9
	3.2 Device Access Functions.....	11
	3.2.1 tdrv012Read.....	11
	3.2.2 tdrv012WriteMask.....	13
	3.2.3 tdrv012OutputSet.....	15
	3.2.4 tdrv012OutputClear.....	17
	3.2.5 tdrv012ConfigureDirection.....	19
	3.2.6 tdrv012ReadDirection.....	21
	3.2.7 tdrv012WaitEvent.....	23
	3.2.8 tdrv012WaitHigh.....	26
	3.2.9 tdrv012WaitLow.....	28
	3.2.10 tdrv012WaitAny.....	30
4	DEVICE DRIVER PROGRAMMING.....	32
	4.1 TDRV012 Files and I/O Functions.....	32
	4.1.1 Opening a Device.....	32
	4.1.2 Closing a Device.....	34
	4.1.3 TDRV012 Device I/O Control Functions.....	35
	4.1.3.1 IOCTL_TDRV012_WRITE.....	37
	4.1.3.2 IOCTL_TDRV012_READ.....	39
	4.1.3.3 IOCTL_TDRV012_OUTPUTENABLE.....	41
	4.1.3.4 IOCTL_TDRV012_GET_DIRECTION.....	43
	4.1.3.5 IOCTL_TDRV012_EVENTWAIT.....	45

1 Introduction

The TDRV012-SW-65 Windows device driver is a kernel mode driver which allows the operation of the supported hardware module on an Intel or Intel-compatible Windows operating system. Supported Windows versions are:

- Windows 2000
- Windows XP
- Windows XP Embedded
- Windows 7 (32bit and 64bit)

The standard file and device (I/O) functions (CreateFile, CloseHandle, and DeviceIoControl) provide the basic interface for opening and closing a resource handle and for performing device I/O control operations.

The TDRV012-SW-65 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-65 device driver 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 TDRV012 devices it is recommended to read the manuals listed below.

TPMC683 User Manual
TPMC683 Engineering Manual

2 Installation

Following files are located in directory TDRV012-SW-65 on the distribution media:

i386\	Directory containing driver files for 32bit Windows versions
amd64\	Directory containing driver files for 64bit Windows versions
installer_32bit.exe	Installation tool for 32bit systems (Windows XP or later)
installer_64bit.exe	Installation tool for 64bit systems (Windows XP or later)
tdrv012.inf	Windows installation script
tdrv012.h	Header file with IOCTL codes and structure definitions
example\tdrv012exa.c	Example application
api\tdrv012api.c	Application Programming Interface source
api\tdrv012api.h	Application Programming Interface header
TDRV012-SW-65-2.0.0.pdf	This document
Release.txt	Information about the Device Driver Release
ChangeLog.txt	Release history

2.1 Software Installation

2.1.1 Windows 2000 / XP

This section describes how to install the TDRV012 Device Driver on a Windows 2000 / XP operating system.

After installing the TDRV012 card(s) and boot-up your system, Windows 2000 / XP setup will show a "**New hardware found**" dialog box.

1. The "**Upgrade Device Driver Wizard**" dialog box will appear on your screen. Click "**Next**" button to continue.
2. In the following dialog box, choose "**Search for a suitable driver for my device**". Click "**Next**" button to continue.
3. Insert the TDRV012 driver media; select "**Disk Drive**" in the dialog box. Click "**Next**" button to continue.
4. Now the driver wizard should find a suitable device driver on the media. Click "**Next**" button to continue.
5. Complete the upgrade device driver and click "**Finish**" to take all the changes effect.
6. Now copy all needed files (tdrv012.h and API files) to the desired target directories.

After successful installation the TDRV012 device driver will start immediately and creates devices (TDRV012_1, TDRV012_2 ...) for all recognized TDRV012 modules.

2.1.2 Windows 7

This section describes how to install the TDRV012-SW-65 Device Driver on a Windows 7 (32bit or 64bit) operating system.

Depending on the operating system type, execute the installer binaries for either 32bit or 64bit systems. This will install all required driver files using an installation wizard.

Copy needed files (tdrv012.h and API files) to desired target directory.

After successful installation a device is created for each module found (TDRV012_1, TDRV012_2 ...).

2.2 Confirming Driver Installation

To confirm that the driver has been properly loaded, perform the following steps:

1. Open the Windows Device Manager:
 - a. For Windows 2000 / XP, open the "**Control Panel**" from "**My Computer**" and click the "**System**" icon and choose the "**Hardware**" tab, and then click the "**Device Manager**" button.
 - b. For Windows 7, open the "**Control Panel**" from "**My Computer**" and then click the "**Device Manager**" entry.
2. Click the "+" in front of "**Embedded I/O**".
The driver "**TEWS TECHNOLOGIES – TDRV012 (Digital I/O) (TPMC683)**" should appear for each installed device.

3 API Documentation

3.1 General Functions

3.1.1 tdrv012Open

NAME

tdrv012Open – Opens 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.

EXAMPLE

```
#include "tdrv012api.h"

TDRV012_HANDLE hdl;

/*
** open file descriptor to device
*/
hdl = tdrv012Open("\\\\.\\TDRV012_1" );
if (hdl == NULL)
{
    /* handle open error */
}
```

RETURNS

A device handle, or NULL if the function fails. To get extended error information, call **GetLastError**.

ERROR CODES

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

3.1.2 tdrv012Close

NAME

tdrv012Close – Closes 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

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,
    uint32_t        *pIoValue
);
```

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.

pIoValue

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;
uint32_t          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

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,
    uint32_t      OutputValue,
    uint32_t      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

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,
    uint32_t      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

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,
    uint32_t      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

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,
    uint32_t        DirectionValue,
    uint32_t        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

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,
    uint32_t      *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 a uint32_t 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;
uint32_t        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

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,
    uint32_t       mask_high,
    uint32_t       mask_low,
    int            timeout,
    uint32_t       *pIoValue,
    uint32_t       *pStatusHigh,
    uint32_t       *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, although the granularity is in seconds. Use -1 to wait indefinitely for the event.

pIoValue

This value is a pointer to a `uint32_t` 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 a `uint32_t` 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 a `uint32_t` 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;
uint32_t 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%08lX\n", IoValue);
    printf(" HIGH transition status: 0x%08lX\n", StatusHigh);
    printf(" LOW  transition status: 0x%08lX\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

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,
    uint32_t mask,
    int timeout,
    uint32_t *pIoValue,
    uint32_t *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, although the granularity is in seconds. Use -1 to wait indefinitely for the event.

pIoValue

This value is a pointer to a uint32_t 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 a uint32_t 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;
uint32_t          IoValue;
uint32_t          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

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,
    uint32_t mask,
    int timeout,
    uint32_t *pIoValue,
    uint32_t *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, although the granularity is in seconds. Use -1 to wait indefinitely for the event.

pIoValue

This value is a pointer to a uint32_t 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 a uint32_t 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;
uint32_t          IoValue;
uint32_t          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

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,
    uint32_t mask,
    int timeout,
    uint32_t *pIoValue,
    uint32_t *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 a uint32_t 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 a uint32_t 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;
uint32_t          IoValue;
uint32_t          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

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 Device Driver Programming

The TDRV012-SW-65 Windows WDM device driver is a kernel mode device driver.

The standard file and device (I/O) functions (CreateFile, CloseHandle, and DeviceIoControl) provide the basic interface for opening and closing a resource handle and for performing device I/O control operations.

All of these standard Win32 functions are described in detail in the Windows Platform SDK Documentation (Windows base services / Hardware / Device Input and Output).

For details refer to the Win32 Programmers Reference of your used programming tools (C++, Visual Basic etc.)

4.1 TDRV012 Files and I/O Functions

The following section does not contain a full description of the Win32 functions for interaction with the TDRV012 device driver. Only the required parameters are described in detail.

4.1.1 Opening a Device

Before you can perform any I/O the *TDRV012* device must be opened by invoking the **CreateFile** function. **CreateFile** returns a handle that can be used to access the *TDRV012* device.

```
HANDLE CreateFile(
    LPCTSTR                lpFileName,
    DWORD                 dwDesiredAccess,
    DWORD                 dwShareMode,
    LPSECURITY_ATTRIBUTES lpSecurityAttributes,
    DWORD                 dwCreationDisposition,
    DWORD                 dwFlagsAndAttributes,
    HANDLE                hTemplateFile
);
```

PARAMETERS

lpFileName

This parameter points to a null-terminated string, which specifies the name of the TDRV012 to open. The *lpFileName* string should be of the form `\\.\TDRV012_x` to open the device *x*. The ending *x* is a one-based number. The first device found by the driver is `\\.\TDRV012_1`, the second `\\.\TDRV012_2` and so on.

dwDesiredAccess

This parameter specifies the type of access to the TDRV012. For the TDRV012 this parameter must be set to read-write access (GENERIC_READ | GENERIC_WRITE)

dwShareMode

Set of bit flags that specify how the object can be shared. Set to 0.

lpSecurityAttributes

This argument is a pointer to a security structure. Set to NULL for TDRV012 devices.

dwCreationDistribution

Specifies the action to take on existing files, and which action to take when files do not exist. TDRV012 devices must be always opened **OPEN_EXISTING**.

dwFlagsAndAttributes

Specifies the file attributes and flags for the file. This value must be set to 0 (no overlapped I/O).

hTemplateFile

This value must be NULL for TDRV012 devices.

RETURN VALUE

If the function succeeds, the return value is an open handle to the specified TDRV012 device. If the function fails, the return value is `INVALID_HANDLE_VALUE`. To get extended error information, call **GetLastError**.

EXAMPLE

```
HANDLE    hDevice;

hDevice = CreateFile(
    "\\.\TDRV012_1",
    GENERIC_READ | GENERIC_WRITE,
    0,
    NULL,           // no security attrs
    OPEN_EXISTING, // TDRV012 device always open existing
    0,             // no overlapped I/O
    NULL
);

if (hDevice == INVALID_HANDLE_VALUE) {
    ErrorHandler( "Could not open device" ); // process error
}
```

SEE ALSO

CloseHandle(), Win32 documentation CreateFile()

4.1.2 Closing a Device

The **CloseHandle** function closes an open TDRV012 handle.

```
BOOL CloseHandle(  
    HANDLE hDevice;  
);
```

PARAMETERS

BOOLEAN hDevice
Identifies an open TDRV012 handle.

RETURN VALUE

If the function succeeds, the return value is nonzero (TRUE).

If the function fails, the return value is zero (FALSE). To get extended error information, call **GetLastError**.

EXAMPLE

```
HANDLE    hDevice;  
  
if( !CloseHandle( hDevice ) ) {  
    ErrorHandler( "Could not close device" ); // process error  
}
```

SEE ALSO

CreateFile (), Win32 documentation CloseHandle ()

4.1.3 TDRV012 Device I/O Control Functions

The **DeviceIoControl** function sends a control code directly to a specified device driver, causing the corresponding device to perform the specified operation.

```

BOOL DeviceIoControl(
    HANDLE                hDevice,
    DWORD                dwIoControlCode,
    LPVOID                lpInBuffer,
    DWORD                nInBufferSize,
    LPVOID                lpOutBuffer,
    DWORD                nOutBufferSize,
    LPDWORD               lpBytesReturned,
    LPOVERLAPPED         lpOverlapped
);
    
```

PARAMETERS

hDevice

Handle to the TDRV012 that is to perform the operation.

dwIoControlCode

This parameter specifies the control code for the operation. This value identifies the specific operation to be performed. The following values are defined in `tdrv012.h`:

Value	Meaning
IOCTL_TDRV012_WRITE	Write output port
IOCTL_TDRV012_READ	Read input port immediately
IOCTL_TDRV012_OUTPUT_ENABLE	Configure input/output direction of I/O lines
IOCTL_TDRV012_GET_DIRECTION	Read current input/output direction configuration
IOCTL_TDRV012_EVENTWAIT	Wait for a specified event

See behind for more detailed information on each control code.

To use these TDRV012 specific control codes the header file `tdrv012.h` must be included in the application.

lpInBuffer

Pointer to a buffer that contains the data required to perform the operation.

nInBufferSize

This argument specifies the size, in bytes, of the buffer pointed to by *lpInBuffer*.

lpOutBuffer

Pointer to a buffer that receives the operation's output data.

nOutBufferSize

This argument specifies the size, in bytes, of the buffer pointed to by *lpOutBuffer*.

lpBytesReturned

Pointer to a variable that receives the size, in bytes, of the data stored into the buffer pointed to by *lpOutBuffer*. A valid pointer is required.

lpOverlapped

This argument is a pointer to an *Overlapped* structure. This value must be set to NULL (no overlapped I/O).

RETURN VALUE

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call ***GetLastError***.

SEE ALSO

Win32 documentation DeviceIoControl ()

4.1.3.1 IOCTL_TDRV012_WRITE

This control function writes the specified value to the output port of the TDRV012 device associated with the open device handle.

The new port value is passed in a buffer (*TDRV012_IOBUFFER*) pointed to by *lpInBuffer*, to the driver. The argument *nInBufferSize* specifies the size of the buffer.

```
typedef struct
{
    uint32_t  value;
    uint32_t  mask;
} TDRV012_IOBUFFER;
```

value

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

mask

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 "tdrv012.h"

HANDLE          hDevice;
BOOLEAN         success;
ULONG           NumBytes;
TDRV012_IOBUFFER outBuf;

/* set I/O lines 1,6,31 to HIGH, 0,2-5 and 7 to LOW */
/* all other I/O lines shall be left unchanged */
outBuf.mask     = 0x800000FF;
outBuf.value    = 0x80000042;

success = DeviceIoControl (
    hDevice,                // device handle
    IOCTL_TDRV012_WRITE,   // control code
    &outBuf,
    sizeof(TDRV012_IOBUFFER),
    NULL,
    0,
    &NumBytes,
    NULL                    // not overlapped
);
...
```

```
...
if( success ) {
    printf("Write output value successful\n");
}
else {
    ErrorHandler("Device I/O control error");
}
```

ERROR CODES

ERROR_INVALID_USER_BUFFER	This error is returned if the size of the user buffer is too small.
---------------------------	---

All other returned error codes are system error conditions.

4.1.3.2 IOCTL_TDRV012_READ

This control function reads the value of the input register of the TDRV012 device associated with the open device handle.

The port value is returned in a buffer (*TDRV012_IOBUFFER*) pointed to by *lpOutBuffer*. The argument *nOutBufferSize* specifies the size of the buffer.

```
typedef struct
{
    uint32_t  value;
    uint32_t  mask;
} TDRV012_IOBUFFER;
```

value

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

mask

This parameter is not used by this function.

EXAMPLE

```
#include "tdrv012.h"

HANDLE          hDevice;
BOOLEAN         success;
ULONG           NumBytes;
TDRV012_IOBUFFER inBuf;

success = DeviceIoControl (
    hDevice,          // device handle
    IOCTL_TDRV012_READ, // control code
    NULL,
    0,
    &inBuf,
    sizeof(TDRV012_IOBUFFER),
    &NumBytes,
    NULL              // not overlapped
);

if( success ) {
    printf("Read input value successful\n");
    printf("    Input value: %08Xh\n", inBuf.value);
}
else {
    ErrorHandler("Device I/O control error");
}
```

ERROR CODES

ERROR_INVALID_USER_BUFFER	This error is returned if the size of the user buffer is too small.
---------------------------	---

All other returned error codes are system error conditions.

4.1.3.3 IOCTL_TDRV012_OUTPUTENABLE

This control function configures the input/output direction of the I/O lines of the TDRV012 device associated with the open device handle.

The new port direction is passed in a buffer (*TDRV012_IOBUFFER*) pointed to by *lpInBuffer*, to the driver. The argument *nInBufferSize* specifies the size of the buffer.

```
typedef struct
{
    uint32_t  value;
    uint32_t  mask;
} TDRV012_IOBUFFER;
```

value

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.

mask

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 "tdrv012.h"

HANDLE          hDevice;
BOOLEAN         success;
ULONG          NumBytes;
TDRV012_IOBUFFER dirBuf;

/*
** 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.
*/
dirBuf.value = (0x00 << 24) | (0xff << 0);
dirBuf.mask  = (0xff << 24) | (0xff << 0);

...
```

...

```
success = DeviceIoControl (
    hDevice,                // device handle
    IOCTL_TDRV012_OUTPUTENABLE, // control code
    &dirBuf,
    sizeof(TDRV012_IOBUFFER),
    NULL,
    0,
    &NumBytes,
    NULL                    // not overlapped
);

if( success ) {
    printf("Configure I/O direction successful\n");
}
else {
    ErrorHandler("Device I/O control error");
}
```

ERROR CODES

ERROR_INVALID_USER_BUFFER	This error is returned if the size of the user buffer is too small.
---------------------------	---

All other returned error codes are system error conditions.

4.1.3.4 IOCTL_TDRV012_GET_DIRECTION

This control function reads the current direction configuration (input/output) of the I/O lines of the TDRV012 device associated with the open device handle.

The direction configuration is returned in a buffer (*TDRV012_IOBUFFER*) pointed to by *lpOutBuffer*. The argument *nOutBufferSize* specifies the size of the buffer.

```
typedef struct
{
    uint32_t  value;
    uint32_t  mask;
} TDRV012_IOBUFFER;
```

value

This value returns 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.

mask

This parameter is not used by this function.

EXAMPLE

```
#include "tdrv012.h"

HANDLE          hDevice;
BOOLEAN         success;
ULONG           NumBytes;
TDRV012_IOBUFFER inBuf;

success = DeviceIoControl (
    hDevice,                // device handle
    IOCTL_TDRV012_GET_DIRECTION, // control code
    NULL,
    0,
    &inBuf,
    sizeof(TDRV012_IOBUFFER),
    &NumBytes,
    NULL                    // not overlapped
);

if( success ) {
    printf("Read direction configuration successful\n");
    printf(" I/O direction (0=INPUT, 1=OUTPUT): %08Xh\n", dirBuf.value);
} else {
    ErrorHandler("Device I/O control error");
}
```

ERROR CODES

ERROR_INVALID_USER_BUFFER	This error is returned if the size of the user buffer is too small.
---------------------------	---

All other returned error codes are system error conditions.

4.1.3.5 IOCTL_TDRV012_EVENTWAIT

This control function waits for an event using the TDRV012 device associated with the open device handle. The function blocks until at least one of the specified events or a timeout occurs.

The event parameters are passed in a buffer (*TDRV012_EVENTWAITBUFFER*) pointed to by *lpInBuffer* and *lpOutBuffer* to the driver. The arguments *nInBufferSize* and *nOutBufferSize* specify the size of the buffer.

```
typedef struct
{
    uint32_t    mask_high;
    uint32_t    mask_low;
    int         timeout;
    uint32_t    iovalue;
    uint32_t    status_high;
    uint32_t    status_low;
} TDRV012_EVENTWAITBUFFER;
```

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, although the granularity is in seconds. Use -1 to wait indefinitely for the event.

iovalue

This value 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.

status_high

This parameter 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.

status_low

This parameter 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 "tdrv012.h"

HANDLE                hDevice;
BOOLEAN              success;
ULONG                NumBytes;
TDRV012_EVENTWAITBUFFER evBuf;

/*
** Wait at least 1000ms for a HIGH event on I/O line 0
*/
evBuf.timeout        = 1000;
evBuf.mask_high     = (1 << 0);
evBuf.mask_low      = 0;

success = DeviceIoControl (
    hDevice,                // device handle
    IOCTL_TDRV012_EVENTWAIT, // control code
    &evBuf,
    sizeof(TDRV012_EVENTWAITBUFFER),
    &evBuf,
    sizeof(TDRV012_EVENTWAITBUFFER),
    &NumBytes,
    NULL                    // not overlapped
);

if( success ) {
    printf("Specified Event occurred.\n");
    printf("  I/O Value: %08Xh\n", evBuf.iovalue);
} else {
    ErrorHandler("Device I/O control error");
}

...
```

```

...

/*
** Wait at least 5000ms for a HIGH event on I/O lines 0..7 or
**           for a LOW event on I/O lines 24..31
*/
evBuf.timeout      = 5000;
evBuf.mask_high    = (0xff << 0);
evBuf.mask_low     = (0xff << 24);

success = DeviceIoControl (
    hDevice,                // device handle
    IOCTL_TDRV012_EVENTWAIT, // control code
    &evBuf,
    sizeof(TDRV012_EVENTWAITBUFFER),
    &evBuf,
    sizeof(TDRV012_EVENTWAITBUFFER),
    &NumBytes,
    NULL                    // not overlapped
);

if( success ) {
    printf("Specified Event occurred.\n");
    printf("  I/O Value      : %08Xh\n", evBuf.iovalue);
    printf("  HIGH event on: %08Xh\n", evBuf.status_high);
    printf("  LOW  event on: %08Xh\n", evBuf.status_low);
} else {
    ErrorHandler("Device I/O control error");
}

```

ERROR CODES

ERROR_INVALID_USER_BUFFER	This error is returned if the size of the user buffer is too small.
ERROR_BUSY	Too many concurrent wait jobs pending (max. 100)
ERROR_SEM_TIMEOUT	Timeout. None of the specified events occurred.

All other returned error codes are system error conditions.