The Embedded I/O Company

TEWS TECHNOLOGIES

TDRV012-SW-42

VxWorks Device Driver

32 differential I/O Lines with Interrupts

Version 1.0.x

User Manual

Issue 1.0.0

November 2008

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

TEWS TECHNOLOGIES LLC
9190 Double Diamond Parkway,
Suite 127, Reno, NV 89521, USA
www.tews.com
Phone: +1 (775) 850 5830
Fax: +1 (775) 201 0347
e-mail: usasales@tews.com

TDRV012-SW-42

VxWorks Device Driver

32 differential I/O Lines with Interrupts

Supported Modules:
TPMC683

Issue	Description	Date
1.0.0	First Issue	November 25, 2008

Table of Contents

1 Introduction

1.1 Device Driver

The TDRV012-SW-42 VxWorks device driver software allows the operation of the supported modules conforming to the VxWorks 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 TDRV012-SW-42 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-42 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 certain devices 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

TPMC683 Engineering Manual

2 Installation

Following files are located on the distribution media:

Directory path 'TDRV012-SW-42':

tdrv012drv.c	TDRV012 device driver source
tdrv012def.h	TDRV012 driver include file
tdrv012.h	TDRV012 include file for driver and application
tdrv012pci.c	TDRV012 PCI MMU mapping for Intel x86 based targets
tdrv012exa.c	Example application
include/tdhal.h	Hardware dependent interface functions and definitions
TDRV012-SW-42-1.0.0.pdf	PDF copy of this manual
ChangeLog.txt	Release history
Release.txt	Release information

2.1 Include device driver in VxWorks project

For including the TDRV012-SW-42 device driver into a VxWorks project (e.g. Tornado IDE or Workbench) follow the steps below:

(1) Copy the files from the distribution media into a subdirectory in your project path. (For example: ./TDRV012)

(2) Add the device drivers C-files to your project.

(3) Now the driver is included in the project and will be built with the project.

For a more detailed description of the project facility please refer to your VxWorks User's Guide (e.g. Tornado, Workbench, etc.)

2.2 Special installation for Intel x86 based targets

The TDRV012 device driver is fully adapted for Intel x86 based targets. This is done by conditional compilation directives inside the source code and controlled by the VxWorks global defined macro **CPU_FAMILY**. If the content of this macro is equal to *I80X86* special Intel x86 conforming code and function calls will be included.

The second problem for Intel x86 based platforms can't be solved by conditional compilation directives. Due to the fact that some Intel x86 BSP's doesn't map PCI memory spaces of devices which are not used by the BSP, the required device memory spaces can't be accessed.

To solve this problem a MMU mapping entry has to be added for the required TDRV012 PCI memory spaces prior the MMU initialization (*usrMmuInit()*) is done.

The C source file **tdrv012pci.c** contains the function *tdrv012PciInit().* This routine finds out all TDRV012 devices and adds MMU mapping entries for all used PCI memory spaces. Please insert a call to this function after the PCI initialization is done and prior to MMU initialization (*usrMmuInit()*).

The right place to call the function *tdrv012PciInit()* is at the end of the function *sysHwInit()* in **sysLib.c** (it can be opened from the project *Files* window).

Be sure that the function is called prior to MMU initialization otherwise the TDRV012 PCI spaces remains unmapped and an access fault occurs during driver initialization.

Please insert the following call at a suitable place in **sysLib.c**:

```
tdrv012PciInit();
```

Modifying the sysLib.c file will change the sysLib.c in the BSP path. Remember this for future projects and recompilations.

2.3 BSP dependent adjustments

The driver includes a file called *include/tdhal.h* which contains functions and definitions for BSP adaptation. It may be necessary to modify them for BSP specific settings. Most settings can be made automatically by conditional compilation set by the BSP header files, but some settings must be configured manually. There are two way of modification, first you can change the *include/tdhal.h* and define the corresponding definition and its value, or you can do it, using the command line option *–D*.

There are 3 offset definitions (*USERDEFINED_MEM_OFFSET*, *USERDEFINED_IO_OFFSET*, and *USERDEFINED_LEV2VEC*) that must be configured if a corresponding warning message appears during compilation. These definitions always need values. Definition values can be assigned by command line option *-D<definition>=<value>*.

definition	**description**
USERDEFINED_MEM_OFFSET	The value of this definition must be set to the offset between CPU-Bus and PCI-Bus Address for PCI memory space access
USERDEFINED_IO_OFFSET	The value of this definition must be set to the offset between CPU-Bus and PCI-Bus Address for PCI I/O space access
USERDEFINED_LEV2VEC	The value of this definition must be set to the difference of the interrupt vector (used to connect the ISR) and the interrupt level (stored to the PCI header)

Another definition allows a simple adaptation for BSPs that utilize a *pciIntConnect()* function to connect shared (PCI) interrupts. If this function is defined in the used BSP, the definition of *USERDEFINED_SEL_PCIINTCONNECT* should be enabled. The definition by command line option is made by *-D<definition>.*

Please refer to the BSP documentation and header files to get information about the interrupt connection function and the required offset values.

2.4 System resource requirement

The table gives an overview over the system resources that will be needed by the driver.

Resource	Driver requirement	Devices requirement
Memory	< 1 KB	< 1 KB
Stack	< 1 KB	---
Semaphores	---	number of jobs

Memory and Stack usage may differ from system to system, depending on the used compiler and its setup.

The following formula shows the way to calculate the common requirements of the driver and devices.

*<total requirement> = <driver requirement> + (<number of devices> * <device requirement>)*

The maximum usage of some resources is limited by adjustable parameters. If the application and driver exceed these limits, increase the according values in your project.

3 I/O system functions

This chapter describes the driver-level interface to the I/O system. The purpose of these functions is to install the driver in the I/O system, add and initialize devices.

3.1 tdrv012Drv()

NAME

tdrv012Drv() - installs the TDRV012 driver in the I/O system

SYNOPSIS

#include “tdrv012.h”

STATUS tdrv012Drv(void)

DESCRIPTION

This function searches for devices on the PCI bus, and installs the TDRV012 driver in the I/O system.

A call to this function is the first thing the user has to do before adding any device to the system or performing any I/O request.

EXAMPLE

```
#include "tdrv012.h”

STATUS             result;

/*-------------------
  Initialize Driver
  -------------------*/
result = tdrv012Drv();
if (result == ERROR)
{
     /* Error handling */
}
```

RETURNS

OK or ERROR. If the function fails an error code will be stored in *errno*.

ERROR CODES

The error codes are stored in *errno* and can be read with the function *errnoGet()*.

Error code	Description
ENOBUFS	Not enough memory for driver requirements.
ENXIO	No devices found

SEE ALSO

VxWorks Programmer's Guide: I/O System

3.2 tdrv012DevCreate()

NAME

tdrv012DevCreate() – Add a TDRV012 device to the VxWorks system

SYNOPSIS

```
#include "tdrv012.h"

STATUS tdrv012DevCreate
(
        char        *name,
        int         devIdx,
        int         funcType,
        void        *pParam
)
```

DESCRIPTION

This routine adds the selected device to the VxWorks system. The device hardware will be setup and prepared for use.

This function must be called before performing any I/O request to this device.

PARAMETER

name

This string specifies the name of the device that will be used to identify the device, for example for *open()* calls.

devIdx

This index number specifies the device to add to the system.

funcType

This parameter is unused and should be set to *0*.

pParam

This parameter is unused and should be set to *NULL*.

EXAMPLE

```
#include "tdrv012.h”

STATUS             result;

/*-------------------------------------------------------
  Create the device "/tdrv012/0" for the first I/O device
  -------------------------------------------------------*/

result = tdrv012DevCreate(   "/tdrv012/0",
                             0,
                             0,
                             NULL);
if (result == OK)
{
     /* Device successfully created */
}
else
{
     /* Error occurred when creating the device */
}
```

RETURNS

OK or ERROR. If the function fails an error code will be stored in *errno*.

ERROR CODES

The error codes are stored in *errno* and can be read with the function *errnoGet()*.

Error code	Description
ENXIO	No matching device found
EBUSY	Device has already been created

SEE ALSO

VxWorks Programmer’s Guide: I/O System

3.3 tdrv012PciInit()

NAME

tdrv012PciInit() – Generic PCI device initialization

SYNOPSIS

void tdrv012PciInit()

DESCRIPTION

This function is required only for Intel x86 VxWorks platforms. The purpose is to setup the MMU mapping for all required TPMC012 PCI spaces (base address register) and to enable the TDRV012 device for access.

The global variable *tdrv012Status* obtains the result of the device initialization and can be polled later by the application before the driver will be installed.

Value	Meaning
> 0	Initialization successful completed. The value of tdrv012Status is equal to the number of mapped PCI spaces
0	No TDRV012 device found
< 0	Initialization failed. The value of (tdrv012Status & 0xFF) is equal to the number of mapped spaces until the error occurs. Possible cause: Too few entries for dynamic mappings in sysPhysMemDesc[]. Remedy: Add dummy entries as necessary (syslib.c).

EXAMPLE

```
extern void tdrv012PciInit();

…

tdrv012PciInit();

…
```

4 I/O Functions

4.1 open()

NAME

open() - open a device or file.

SYNOPSIS

```
int open
(
        const char   *name,
        int          flags,
        int          mode
)
```

DESCRIPTION

Before I/O can be performed to the TDRV012 device, a file descriptor must be opened by invoking the basic I/O function *open().*

PARAMETER

name

Specifies the device which shall be opened, the name specified in tdrv012DevCreate() must be used

flags

Not used

mode

Not used

EXAMPLE

```
int      fd;

/*-----------------------------------------
  Open the device named "/tdrv012/0" for I/O
  -----------------------------------------*/
fd = open("/tdrv012/0", 0, 0);
if (fd == ERROR)
{
     /* Handle error */
}
```

RETURNS

A device descriptor number or ERROR. If the function fails an error code will be stored in *errno*.

ERROR CODES

The error code can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual.

SEE ALSO

ioLib, basic I/O routine - *open()*

4.2 close()

NAME

close() – close a device or file

SYNOPSIS

```
STATUS close
(
    int         fd
)
```

DESCRIPTION

This function closes opened devices.

PARAMETER

fd

This file descriptor specifies the device to be closed. The file descriptor has been returned by the *open()* function.

EXAMPLE

```
int       fd;
STATUS    retval;

/*----------------
  close the device
  ----------------*/
retval = close(fd);
if (retval == ERROR)
{
     /* Handle error */
}
```

RETURNS

OK or ERROR. If the function fails, an error code will be stored in *errno*.

ERROR CODES

The error code can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual)

SEE ALSO

ioLib, basic I/O routine - close()

4.3 ioctl()

NAME

ioctl() - performs an I/O control function.

SYNOPSIS

```
#include “tdrv012.h”

int ioctl
(
        int     fd,
        int     request,
        int     arg
)
```

DESCRIPTION

Special I/O operation that do not fit to the standard basic I/O calls (read, write) will be performed by calling the ioctl() function.

PARAMETER

fd

This file descriptor specifies the device to be used. The file descriptor has been returned by the *open()* function.

request

This argument specifies the function that shall be executed. Following functions are defined:

Function	Description
FIO_TDRV012_READ	Read value of I/O lines
FIO_TDRV012_WRITE_MASK	Write value of specific output lines
FIO_TDRV012_OUTPUT_ENABLE	Enable output (configure I/O line direction)
FIO_TDRV012_EVENT_WAIT	Wait for I/O transition events

arg

This parameter depends on the selected function (request). How to use this parameter is described below with the function.

RETURNS

OK or ERROR. If the function fails an error code will be stored in *errno*.

ERROR CODES

The error code can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual). Function specific error codes will be described with the function.

Error code	Description
ENOSYS	Invalid request value specified

SEE ALSO

ioLib, basic I/O routine - ioctl()

4.3.1 FIO_TDRV012_READ

This I/O control function reads the current value of the Input and Output lines. The function specific control parameter *arg* is a pointer on a *TDRV012_IOBUFFER* structure.

```
typedef struct
{
        unsigned long           value;
        unsigned long           mask;
} TDRV012_IOBUFFER;
```

value

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

mask

This argument returns the direction of the corresponding I/O line. A set bit indicates that the corresponding line is configured as output, an unset bit indicates an input line.

EXAMPLE

```
#include “tdrv012.h”

int                     fd;
TDRV012_IOBUFFER        rBuf;
int                     retval;

/*--------------
  read I/O value
  --------------*/
retval = ioctl(fd, FIO_TDRV012_READ, (int)&rBuf);
if (retval != ERROR)
{
     /* function succeeded */
     printf(“I/O-value:   %08lXh\n”, rBuf.value);
     printf(“Output mask: %08lXh\n”, rBuf.mask);
}
else
{
     /* handle the error */
}
```

4.3.2 FIO_TDRV012_WRITE_MASK

This I/O control function sets the value of the output lines. The function specific control parameter *arg* is a pointer on a *TDRV012_IOBUFFER* structure.

```
typedef struct
{
        unsigned long           value;
        unsigned long           mask;
} TDRV012_IOBUFFER;
```

value

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

mask

This argument specifies a bitmask to define which output lines should be written.

EXAMPLE

```
#include “tdrv012.h”

int                     fd;
TDRV012_IOBUFFER        wBuf;
int                     retval;

/*-----------------------------------------------------
  set I/O line 0-7, leave all other outputs unchanged
  -----------------------------------------------------*/
wBuf.value    = 0x00000042;
wBuf.mask     = 0x000000FF;
retval = ioctl(fd, FIO_TDRV012_WRITE_MASK, (int)&wBuf);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

4.3.3 FIO_TDRV012_OUTPUT_ENABLE

This I/O control function configures the direction of the I/O lines. The function specific control parameter *arg* is a pointer on a *TDRV012_IOBUFFER* structure.

```
typedef struct
{
        unsigned long           value;
        unsigned long           mask;
} TDRV012_IOBUFFER;
```

value

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

mask

This argument specifies a bitmask to define which output lines should be affected.

EXAMPLE

```
#include “tdrv012.h”

int                     fd;
TDRV012_IOBUFFER        dBuf;
int                     retval;

/*-----------------------------------------------
  configure line 0-7 for input, 8-15 for output.
  leave other I/O lines unchanged.
  -----------------------------------------------*/
dBuf.value    = 0x0000FF00;
dBuf.mask     = 0x0000FFFF;
retval = ioctl(fd, FIO_TDRV012_OUTPUT_ENABLE, (int)&dBuf);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

4.3.4 FIO_TDRV012_EVENT_WAIT

This I/O control function waits for an I/O line transition event. The function specific control parameter *arg* is a pointer on a *TDRV012_EVENTWAITBUFFER* structure.

```
typedef struct
{
        unsigned long       mask_high;
        unsigned long       mask_low;
        unsigned long       timeout;
        unsigned long       inputvalue;
        unsigned long       status_high;
        unsigned long       status_low;
} TDRV012_EVENTWAITBUFFER;
```

mask_high

This argument specifies the input lines to be monitored for HIGH transitions. 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.

mask_low

This argument specifies the input lines to be monitored for LOW transitions. 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.

timeout

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

inputvalue

This parameter 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.

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”

int                          fd;
TDRV012_EVENTWAITBUFFER      evBuf;
int                          retval, line;

/*-------------------------------------
  Wait for HIGH transitions on I/O lines 0 and 1,
  Wait for LOW  transitions on I/O lines 2 and 3,
  Wait for ANY  transition  on I/O line 31,
  -------------------------------------*/
evBuf.mask_high    = (1 << 31) | (1 << 1) | (1 << 0);
evBuf.mask_low     = (1 << 31) | (1 << 3) | (1 << 2);
evBuf.timeout      = 600;        /* Ticks */
retval = ioctl(fd, FIO_TDRV012_EVENT_WAIT, (int)&evBuf);
if (retval != ERROR)
{
     /* function succeeded */
     for (line=0; line<32; line++)
     {
          if (evBuf.mask_high & (1 << line))
          {
               printf(“Line %d: HIGH transition\n”, line);
          }
          if (evBuf.mask_low & (1 << line))
          {
               printf(“Line %d: LOW transition\n”, line);
          }
     }
} else {
     /* handle the error */
}
```

ERROR CODES

EINVAL	Invalid parameter specified
EBUSY	An other task is already waiting for a transition of the specified input line

5 API Documentation

5.1 General Functions

5.1.1 tdrv012open()

Name

tdrv012open() – opens a device.

Synopsis

```
int 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 same device name as specified in tdrv012DevCreate() must be used.

Example

```
#include “tdrv012api.h”
int FileDescriptor;

/*
** open file descriptor to device
*/
FileDescriptor = tdrv012open( “/tdrv012/0” );
if (FileDescriptor == ERROR)
{
          /* handle open error */
}
```

RETURNS

A device descriptor number, or ERROR if the function fails. An error code will be stored in *errno*.

ERROR CODES

The error codes are stored in *errno.*

The error code is a standard error code set by the I/O system.

5.1.2 tdrv012close()

Name

tdrv012close() – closes a device.

Synopsis

```
int tdrv012close
(
     int  FileDescriptor
);
```

Description

This function closes previously opened devices.

Parameters

FileDescriptor

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

Example

```
#include “tdrv012api.h”
int FileDescriptor;
int result;

/*
** close file descriptor to device
*/
result = tdrv012close( FileDescriptor );
if (result == ERROR)
{
     /* handle close error */
}
```

RETURNS

OK, or ERROR if the function fails. An error code will be stored in *errno*.

ERROR CODES

The error codes are stored in *errno*.

The error code is a standard error code set by the I/O system.

5.2 Device Access Functions

5.2.1 tdrv012read()

Name

tdrv012read() – read current I/O value.

Synopsis

```
int tdrv012read
(
     int                FileDescriptor,
     unsigned long      *pIoValue
);
```

Description

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

Parameters

FileDescriptor

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

pIoValue

This value is a pointer to an unsigned long 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”
int           FileDescriptor;
int           result;
unsigned long IoValue;

/*
** read current I/O value
*/
result = tdrv012read( FileDescriptor, &IoValue );
if (result == OK)
{
     printf( “I/O Value: 0x%08lX\n”, IoValue );
} else {
     /* handle error */
}
```

RETURNS

On success, OK is returned. In the case of an error, ERROR is returned.

ERROR CODES

The error code can be read with the function *errnoGet()*.

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

5.2.2 tdrv012writeMask()

Name

tdrv012writeMask() – write relevant bits of output value.

Synopsis

```
int tdrv012writeMask
(
     int                FileDescriptor,
     unsigned long      OutputValue,
     unsigned long      BitMask
);
```

Description

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

Parameters

FileDescriptor

This value specifies the file descriptor 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”
int            FileDescriptor;
int            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(
               FileDescriptor,
               (1 << 1),
               (1 << 1) | (1 << 6)
          );
if (result == OK)
{
     /* OK */
} else {
     /* handle error */
}
```

RETURNS

On success, OK is returned. In the case of an error, ERROR is returned.

ERROR CODES

The error code can be read with the function *errnoGet()*.

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

5.2.3 tdrv012outputSet()

Name

tdrv012outputSet() – set single output lines to ON.

Synopsis

```
int tdrv012outputSet
(
     int                FileDescriptor,
     unsigned long      OutputValue
);
```

Description

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

Parameters

FileDescriptor

This value specifies the file descriptor 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”
int            FileDescriptor;
int            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(
               FileDescriptor,
               (1 << 1) | (1 << 2)
          );
if (result == OK)
{
     /* OK */
} else {
     /* handle error */
}
```

RETURNS

On success, OK is returned. In the case of an error, ERROR is returned.

ERROR CODES

The error code can be read with the function *errnoGet()*.

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

5.2.4 tdrv012outputClear()

Name

tdrv012outputClear() – clear single output lines to OFF.

Synopsis

```
int tdrv012outputClear
(
     int                FileDescriptor,
     unsigned long      OutputValue
);
```

Description

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

Parameters

FileDescriptor

This value specifies the file descriptor 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”
int            FileDescriptor;
int            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(
               FileDescriptor,
               (1 << 1) | (1 << 3)
          );
if (result == OK)
{
     /* OK */
} else {
     /* handle error */
}
```

RETURNS

On success, OK is returned. In the case of an error, ERROR is returned.

ERROR CODES

The error code can be read with the function *errnoGet()*.

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

5.2.5 tdrv012configureDirection()

Name

tdrv012configureDirection() – configure input/output direction of I/O lines.

Synopsis

```
int tdrv012configureDirection
(
     int                FileDescriptor,
     unsigned long      DirectionValue,
     unsigned long      DirectionMask
);
```

Description

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

Parameters

FileDescriptor

This value specifies the file descriptor 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. Active (1) bits will configure the corresponding I/O line to OUTPUT, unset (0) bits 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, 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”
int            FileDescriptor;
int            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 = ttdrv012configureDirection(
               FileDescriptor,
               (0x00 << 24) | (0xff << 0),
               (0xff << 24) | (0xff << 0)
          );
if (result == OK)
{
     /* OK */
} else {
     /* handle error */
}
```

RETURNS

On success, OK is returned. In the case of an error, ERROR is returned.

ERROR CODES

The error code can be read with the function *errnoGet()*.

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

5.2.6 tdrv012readDirection()

Name

tdrv012readDirection() – read current input/output direction configuration of I/O lines.

Synopsis

```
int tdrv012readDirection
(
     int                FileDescriptor,
     unsigned long      *pDirectionValue
);
```

Description

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

Parameters

FileDescriptor

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

pDirectionValue

This value is a pointer to an unsigned long 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”
int           FileDescriptor;
int           result;
unsigned long DirectionValue;

/*
** read current I/O direction configuration
*/
result = tdrv012readDirection(
              FileDescriptor,
              &DirectionValue
         );
if (result == OK)
{
     printf(“Current direction configuration (1=OUTPUT, 0=INPUT):\n”);
     printf(“  0x%08lX\n”, DirectionValue);
} else {
     /* handle error */
}
```

RETURNS

On success, OK is returned. In the case of an error, ERROR is returned.

ERROR CODES

The error code can be read with the function *errnoGet()*.

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

5.2.7 tdrv012waitEvent()

Name

tdrv012waitEvent() – wait for specific transitions on I/O lines.

Synopsis

```
int tdrv012waitEvent
(
     int                FileDescriptor,
     unsigned long      mask_high,
     unsigned long      mask_low,
     int                timeout,
     unsigned long      *pIoValue,
     unsigned long      *pStatusHigh,
     unsigned long      *pStatusLow
);
```

Description

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

Parameters

FileDescriptor

This value specifies the file descriptor 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 long 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.

pStatusHigh

This parameter is a pointer to an unsigned long 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 long 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”
int           FileDescriptor;
int           result;
unsigned long 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(
              FileDescriptor,
              (1 << 2) | (1 << 0),
              (1 << 2) | (1 << 1),
              1000,
              &IoValue,
              &StatusHigh,
              &StatusLow
         );
if (result == 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, OK is returned. In the case of an error, ERROR is returned.

ERROR CODES

The error code can be read with the function *errnoGet()*.

EINVAL	Invalid parameter specified
EBUSY	An other task is already waiting for a transition of the specified input line

5.2.8 tdrv012waitHigh()

Name

tdrv012waitHigh() – wait for HIGH transitions on specific I/O lines.

Synopsis

```
int tdrv012waitHigh
(
     int                FileDescriptor,
     unsigned long      mask,
     int                timeout,
     unsigned long      *pIoValue,
     unsigned long      *pStatus
);
```

Description

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

Parameters

FileDescriptor

This value specifies the file descriptor 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 long 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 long 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”
int           FileDescriptor;
int           result;
unsigned long IoValue;
unsigned long Status;

/*
** wait at least 1000ms for HIGH transition events:
** HIGH transition on I/O line 31
*/
result = tdrv012waitHigh(
              FileDescriptor,
              (1 << 31),
              1000,
              &IoValue,
              &Status
         );
if (result == OK)
{
     printf(“  Current I/O status    : 0x%08lX\n”, IoValue);
     printf(“  HIGH transition status: 0x%08lX\n”, Status);
} else {
     /* handle error */
}
```

RETURNS

On success, OK is returned. In the case of an error, ERROR is returned.

ERROR CODES

The error code can be read with the function *errnoGet()*.

EINVAL	Invalid parameter specified
EBUSY	An other task is already waiting for a transition of the specified input line

5.2.9 tdrv012waitLow()

Name

tdrv012waitLow() – wait for LOW transitions on specific I/O lines.

Synopsis

```
int tdrv012waitLow
(
     int                FileDescriptor,
     unsigned long      mask,
     int                timeout,
     unsigned long      *pIoValue,
     unsigned long      *pStatus
);
```

Description

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

Parameters

FileDescriptor

This value specifies the file descriptor 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 long 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 long 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”
int           FileDescriptor;
int           result;
unsigned long IoValue;
unsigned long Status;

/*
** wait at least 1000ms for LOW transition events:
** LOW transition on I/O line 31
*/
result = tdrv012waitLow(
              FileDescriptor,
              (1 << 31),
              1000,
              &IoValue,
              &Status
         );
if (result == OK)
{
     printf(“  Current I/O status   : 0x%08lX\n”, IoValue);
     printf(“  LOW transition status: 0x%08lX\n”, Status);
} else {
     /* handle error */
}
```

RETURNS

On success, OK is returned. In the case of an error, ERROR is returned.

ERROR CODES

The error code can be read with the function *errnoGet()*.

EINVAL	Invalid parameter specified
EBUSY	An other task is already waiting for a transition of the specified input line