The Embedded I/O Company

TEWS TECHNOLOGIES

TDRV012-SW-82

Linux Device Driver

32 differential I/O Lines with Interrupts

Version 1.0.x

User Manual

Issue 1.0.0

March 2009

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-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

Table of Contents

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. This includes a device-independent basic I/O interface with *open()*, *close(),*and *ioctl()* functions.

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

TPMC683 Engineering Manual

2 Installation

The directory TDRV012-SW-82 on the distribution media contains the following files:

TDRV012-SW-82-1.0.0.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:

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

In order to perform an installation, extract all files of the archive TDRV012-SW-82.tar.gz to the desired target directory. Additionally, copy *tdrv012.h* into your include path (/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 Device Input/Output functions

This chapter describes the interface to the device driver I/O system.

3.1 open()

NAME

open() opens a file descriptor.

SYNOPSIS

#include <fcntl.h>

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

DESCRIPTION

The **open** function creates and returns a new file descriptor for the file named by *filename*. The *flags* argument controls how the file is to be opened. This is a bit mask. Create the value by the bitwise OR of the appropriate parameters (using the | operator in C). See also the GNU C Library documentation for more information about the open function and open flags.

EXAMPLE

```
int fd;

fd = open("/dev/tdrv012_0", O_RDWR);
if (fd == -1)
{
   /* handle error condition */
}
```

RETURNS

The normal return value from **open** is a non-negative integer file descriptor. In case of an error, a value of –1 is returned. The global variable *errno* contains the detailed error code.

ERRORS

E_NODEV	The requested minor device does not exist.

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

SEE ALSO

GNU C Library description – Low-Level Input/Output

3.2 close()

NAME

close() closes a file descriptor.

SYNOPSIS

#include <unistd.h>

int **close** (int *filedes*)

DESCRIPTION

The **close** function closes the file descriptor *filedes*.

EXAMPLE

```
int fd;

if (close(fd) != 0)
{
   /* handle close error conditions */
}
```

RETURNS

The normal return value from **close** is 0. In case of an error, a value of –1 is returned. The global variable *errno* contains the detailed error code.

ERRORS

E_NODEV	The requested minor device does not exist.

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

SEE ALSO

GNU C Library description – Low-Level Input/Output

3.3 ioctl()

NAME

ioctl() device control functions

SYNOPSIS

#include <sys/ioctl.h>

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

DESCRIPTION

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

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

The following ioctl codes are defined in *tdrv012.h*:

Value	Meaning
TDRV012_IOCG_READ	Read value of I/O lines
TDRV012_IOCS_WRITE	Write value of specific output lines
TDRV012_IOCS_OUTPUT_ENABLE	Enable output (configure I/O line direction)
TDRV012_IOCR_GET_DIRECTION	Read I/O line direction configuration
TDRV012_IOCX_EVENT_WAIT	Wait for I/O transition events

See below 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.

RETURNS

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

ERRORS

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

Other function dependent error codes will be described for each ioctl code separately. Note, the TDRV012 device driver always returns standard Linux error codes.

SEE ALSO

ioctl man pages

3.3.1 TDRV012_IOCG_READ

NAME

TDRV012_IOCG_READ Read value of I/O lines

DESCRIPTION

This ioctl function reads the current value of the Input and Output lines. A pointer to the caller's data buffer (*TDRV012_IOBUFFER*) is passed to the device driver by the argument *argp*.

```
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;
int                 result;
TDRV012_IOBUFFER    rBuf;

result = ioctl(fd, TDRV012_IOCG_READ, &rBuf);

if (retval >= 0)
{
  /* function succeeded */
  printf("I/O-value:   %08lXh\n", rBuf.value);
  printf("Output mask: %08lXh\n", rBuf.mask);
} else {
  /* handle the error */
}
```

ERRORS

EINVAL	Invalid parameter (NULL pointer)
EFAULT	Error while copying data to or from user space.

Other returned error codes are system error conditions.

3.3.2 TDRV012_IOCS_WRITE

NAME

TDRV012_IOCS_WRITE Write value of specific output lines

DESCRIPTION

This ioctl function sets the value of the output lines. A pointer to the caller's data buffer (*TDRV012_IOBUFFER*) is passed to the device driver by the argument *argp.*

```
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;
int                 result;
TDRV012_IOBUFFER    wBuf;

/*-----------------------------------------------------
  set I/O line 0-7, leave all other outputs unchanged
  -----------------------------------------------------*/
wBuf.value  = 0x00000042;
wBuf.mask   = 0x000000FF;
retval = ioctl(fd, TDRV012_IOCS_WRITE, &wBuf);
if (retval >= 0)
{
   /* function succeeded */
} else {
   /* handle the error */
}
```

ERRORS

EINVAL	Invalid parameter (NULL pointer)
EFAULT	Error while copying data to or from user space.

Other returned error codes are system error conditions.

3.3.3 TDRV012_IOCS_OUTPUT_ENABLE

NAME

TDRV012_IOCS_OUTPUT_ENABLE Enable output (configure I/O line direction)

DESCRIPTION

This ioctl function configures the direction of the I/O lines. A pointer to the caller's data buffer (*TDRV012_IOBUFFER*) is passed to the device driver by the argument *argp.*

```
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;
int                 result;
TDRV012_IOBUFFER    dBuf;

/*-----------------------------------------------
  configure line 0-7 for input, 8-15 for output.
  leave other I/O lines unchanged.
  -----------------------------------------------*/
dBuf.value  = 0x0000FF00;
dBuf.mask   = 0x0000FFFF;

result = ioctl(fd, TDRV012_IOCS_OUTPUT_ENABLE, &dBuf);
if (result < 0)
{
    /* handle ioctl error */
}
```

ERRORS

EINVAL	Invalid parameter (NULL pointer)
EFAULT	Error while copying data to or from user space.

Other returned error codes are system error conditions.

3.3.4 TDRV012_IOCG_GET_DIRECTION

NAME

TDRV012_IOCG_GET_DIRECTION Read I/O line direction configuration

DESCRIPTION

This ioctl function returns the current input/output direction configuration of the I/O lines. A pointer to the caller's data buffer (*TDRV012_IOBUFFER*) is passed to the device driver by the argument *argp.*

```
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. An output line is indicated by a set bit, an unset bit indicates an input (tri-state) line.

mask

This argument is not used by this function.

EXAMPLE

```
#include "tdrv012.h"

int                 fd;
int                 result;
TDRV012_IOBUFFER    dBuf;

/*-----------------------------------------------
  read input/output direction configuration
  -----------------------------------------------*/
result = ioctl(fd, TDRV012_IOCG_GET_DIRECTION, &dBuf);
if (result < 0)
{
    /* handle ioctl error */
} else {
    printf("I/O Direction (1=output, 0=input): %08Xh\n", dBuf.value);
}
```

ERRORS

EINVAL	Invalid parameter (NULL pointer)
EFAULT	Error while copying data to or from user space.

Other returned error codes are system error conditions.

3.3.5 TDRV012_IOCX_EVENT_WAIT

NAME

TDRV012_IOCX_EVENT_WAIT Wait for I/O transition events

DESCRIPTION

This ioctl function waits for an I/O line transition event. A pointer to the caller's data buffer (*TDRV012_EVENTWAITBUFFER*) is passed to the device driver by the argument *argp.*

```
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 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 milliseconds.

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;
int                     result, line;
TDRV012_EVENTWAITBUFFER evBuf;

/*-------------------------------------
  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      = 10000;      /* milliseconds = 10 seconds */
retval = ioctl(fd, TDRV012_IOCX_EVENT_WAIT, &evBuf);
if (retval >= 0)
{
   /* 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 */
}
```

ERRORS

EINVAL	Invalid parameter (NULL pointer)
EBUSY	Too many concurrent wait jobs pending.
ETIME	Timeout. None of the specified events occurred.
EFAULT	Error while copying data to or from user space.

Other returned error codes are system error conditions.

4 API Documentation

4.1 General Functions

4.1.1 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( “/dev/tdrv012_0” );
if (FileDescriptor == -1)
{
     /* handle open error */
}
```

RETURNS

A device descriptor number, or -1 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.

4.1.2 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 == -1)
{
  /* handle close error */
}
```

RETURNS

Zero, or -1 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.

4.2 Device Access Functions

4.2.1 tdrv012read()

Name

tdrv012read() – read current I/O value.

Synopsis

```
int tdrv012read
(
  int         FileDescriptor,
  uint32_t    *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 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”
int       FileDescriptor;
int       result;
uint32_t  IoValue;

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

RETURNS

On success, zero is returned. In the case of an error, the negative error code is returned.

ERROR CODES

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

4.2.2 tdrv012writeMask()

Name

tdrv012writeMask() – write relevant bits of output value.

Synopsis

```
int tdrv012writeMask
(
  int         FileDescriptor,
  uint32_t    OutputValue,
  uint32_t    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 == 0)
{
   /* OK */
} else {
   /* handle error */
}
```

RETURNS

On success, zero is returned. In the case of an error, the negative error code is returned.

ERROR CODES

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

4.2.3 tdrv012outputSet()

Name

tdrv012outputSet() – set single output lines to ON.

Synopsis

```
int tdrv012outputSet
(
  int         FileDescriptor,
  uint32_t    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 == 0)
{
  /* OK */
} else {
  /* handle error */
}
```

RETURNS

On success, zero is returned. In the case of an error, the negative error code is returned.

ERROR CODES

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

4.2.4 tdrv012outputClear()

Name

tdrv012outputClear() – clear single output lines to OFF.

Synopsis

```
int tdrv012outputClear
(
  int         FileDescriptor,
  uint32_t    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 == 0)
{
  /* OK */
} else {
  /* handle error */
}
```

RETURNS

On success, zero is returned. In the case of an error, the negative error code is returned.

ERROR CODES

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

4.2.5 tdrv012configureDirection()

Name

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

Synopsis

```
int tdrv012configureDirection
(
  int         FileDescriptor,
  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

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. 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, 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 == 0)
{
  /* OK */
} else {
  /* handle error */
}
```

RETURNS

On success, zero is returned. In the case of an error, the negative error code is returned.

ERROR CODES

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

4.2.6 tdrv012readDirection()

Name

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

Synopsis

```
int tdrv012readDirection
(
  int         FileDescriptor,
  uint32_t    *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 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”
int       FileDescriptor;
int       result;
uint32_t  DirectionValue;

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

RETURNS

On success, zero is returned. In the case of an error, the negative error code is returned.

ERROR CODES

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

4.2.7 tdrv012waitEvent()

Name

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

Synopsis

```
int tdrv012waitEvent
(
  int         FileDescriptor,
  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

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 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.

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”
int       FileDescriptor;
int       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(
       FileDescriptor,
       (1 << 2) | (1 << 0),
       (1 << 2) | (1 << 1),
       1000,
       &IoValue,
       &StatusHigh,
       &StatusLow
     );
if (result == 0)
{
  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, zero is returned. In the case of an error, the negative error code is returned.

ERROR CODES

EBUSY	Too many concurrent wait jobs pending.
ETIME	Timeout. None of the specified events occurred.

4.2.8 tdrv012waitHigh()

Name

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

Synopsis

```
int tdrv012waitHigh
(
  int         FileDescriptor,
  uint32_t    mask,
  int         timeout,
  uint32_t    *pIoValue,
  uint32_t    *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 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”
int       FileDescriptor;
int       result;
uint32_t  IoValue;
uint32_t  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 == 0)
{
  printf(“  Current I/O status    : 0x%08X\n”, IoValue);
  printf(“  HIGH transition status: 0x%08X\n”, Status);
} else {
  /* handle error */
}
```

RETURNS

On success, zero is returned. In the case of an error, the negative error code is returned.

ERROR CODES

EBUSY	Too many concurrent wait jobs pending.
ETIME	Timeout. None of the specified events occurred.

4.2.9 tdrv012waitLow()

Name

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

Synopsis

```
int tdrv012waitLow
(
  int         FileDescriptor,
  uint32_t    mask,
  int         timeout,
  uint32_t    *pIoValue,
  uint32_t    *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 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”
int       FileDescriptor;
int       result;
uint32_t  IoValue;
uint32_t  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 == 0)
{
  printf(“  Current I/O status   : 0x%08X\n”, IoValue);
  printf(“  LOW transition status: 0x%08X\n”, Status);
} else {
  /* handle error */
}
```

RETURNS

On success, zero is returned. In the case of an error, the negative error code is returned.

ERROR CODES

EBUSY	Too many concurrent wait jobs pending.
ETIME	Timeout. None of the specified events occurred.

5 Diagnostic

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

To get debug output from the driver enable the following symbols in “tdrv012.c” by replacing “#undef” with “#define”, recompile and reinstall the driver:

```
#define DEBUG_TDRV012
```

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).

```
# tail –f /var/log/messages /* before modprobing the driver */
kernel: TEWS TECHNOLOGIES - TDRV012 Device Driver: version 1.0.x (<date>)
kernel:
kernel: TDRV012: Probe new device (vendor=0x1498, device=0x02AB)
udev[5947]: creating device node '/dev/tdrv012_0'
...
```

```
# 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
. . .
```