The Embedded I/O Company

TEWS TECHNOLOGIES

TDRV005-SW-42

VxWorks Device Driver

6 Channel SSI, Incremental Encoder, Counter

Version 2.1.x

User Manual

Issue 2.1.0

February 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

TDRV005-SW-42

VxWorks Device Driver

6 Channel SSI, Incremental Encoder, Counter

Supported Modules:
TPMC117
TPMC317

Issue	Description	Date
1.0.0	First Issue	December 14, 2005
1.0.1	Examples and structures corrected	December 15, 2005
2.0.0	New address TEWS LLC, general revision, user interface modified	July 17, 2008
2.1.0	Support for TPMC317 added	February 12, 2009

Table of Contents

1 Introduction

The TDRV005-SW-42 VxWorks device driver software allows the operation of the TPMC117 family PMC conforming to the VxWorks I/O system specification. This includes a device-independent basic I/O interface with *open*(), *close()* and *ioctl()* functions.

The provided Application Programming Interface (API) should be used to access the TDRV005 specific functions. This API enhances the compatibility of a TDRV005 based application to different operating systems. The API itself makes use of an abstraction layer, which adapts the different operating system entry calls. This results in a compatibility of the API too.

The TDRV005-SW-42 device driver and its API support the following features:

- operate channels in SSI mode
  - setup and configure channel (SSI or SSI listen-only)
  - read SSI data
- operate channels in Counter mode
  - setup and configure channel
  - read counter data
  - setup preload value
  - load preload value into counter
  - reset counter
  - wait for MATCH event
  - wait for ControlMode event
- operate the onboard interval timer
  - setup and configure interval timer
  - start and stop interval timer
  - read interval timer value
  - wait for interval timer event
  - setup and use interval timer as trigger event for simultaneous multiple channel read
- enable and disable multiple channels simultaneously
- simultaneously read multiple channel values
- setup and load counter preload values simultaneously

The TDRV005-SW-42 supports the modules listed below:

TPMC117	6 Channel SSI, Incremental Encoder, Counter	(PMC)
TPMC317	6 Channel SSI, Incremental Encoder, Counter	(PMC, Conduction Cooled

To get more information about the features and use of TDRV005 devices it is recommended to read the manuals listed below.

TPMC117/TPMC317 User manual

TPMC117/TPMC317 Engineering Manual

2 Installation

Following files are located on the distribution media:

Directory path 'TDRV005-SW-42':

tdrv005drv.c	TDRV005 device driver source
tdrv005def.h	TDRV005 driver include file
tcdi/tcdi_lib.h	Device Driver Interface header file (used by device driver)
tcdi/tcdi_lib.c	Device Driver Interface source file
tdrv005.h	TDRV005 include file for driver and application
tdrv005pci.c	TDRV005 PCI MMU mapping for Intel x86 based targets
include/tdhal.h	Hardware dependent interface functions and definitions
tcosi_lib.h	Common Operating System Interface abstraction layer (used by API)
tdrv005api.h	API include file
tdrv005api.c	API source file
tdrv005exa.c	Example application
TDRV005-SW-42-2.1.0.pdf	PDF copy of this manual
Release.txt	Release information
ChangeLog.txt	Release history

2.1 Include device driver in Tornado IDE project

For Including the TDRV005-SW-42 device driver into a Tornado IDE project follow the steps below:

(1) Copy the files and subdirectories from the distribution media into a subdirectory in your project path. (For example: ./TDRV005)

(2) Add the device drivers C-files to your project.
Make a right click to your project in the 'Workspace' window and use the 'Add Files ...' topic. A file select box appears, and the driver files can be selected.

(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 Tornado User's Guide.

2.2 Special installation for Intel x86 based targets

The TDRV005 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 TDRV005 PCI memory spaces prior the MMU initialization (*usrMmuInit()*) is done.

The C source file **tdrv005pci.c** contains the function *tdrv005PciInit().* This routine finds out all TDRV005 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 *tdrv005PciInit()* 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 TDRV005 PCI spaces remain unmapped and an access fault occurs during driver initialization.

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

```
tdrv005PciInit();
```

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

2.3 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	20	---

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 tdrv005Drv()

NAME

tdrv005Drv() - installs the TDRV005 driver in the I/O system

SYNOPSIS

#include “tdrv005.h”

STATUS tdrv005Drv(void)

DESCRIPTION

This function searches for devices on the PCI bus, installs the TDRV005 driver in the I/O system.

The call of 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 "tdrv005.h”

/*-------------------
  Initialize Driver
  -------------------*/
status = tdrv005Drv();
if (status == ERROR)
{
     /* Error handling */
}
```

RETURNS

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

ERROR CODES

Error codes are only set by system functions. The error codes are stored in *errno* and can be read with the function *errnoGet()*.

SEE ALSO

VxWorks Programmer's Guide: I/O System

3.2 tdrv005DevCreate()

NAME

tdrv005DevCreate() – Add a TDRV005 device to the VxWorks system

SYNOPSIS

```
#include “tdrv005.h”

STATUS tdrv005DevCreate
(
    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.
If modules of the same type are installed the channel numbers will be assigned in the order the VxWorks *pciFindDevice()* function will find the devices.

Example: A system with 2x installed TPMC117-xx and 1x TPMC317 modules will assign the following device indexes:

Module	Device Index
TPMC117-xx (1st)	0
TPMC117-xx (2nd)	1
TPMC317-xx	2

funcType

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

pParam

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

EXAMPLE

```
#include "tdrv005.h”

STATUS            result;

/*-------------------------------------------------------
  Create the device "/tdrv005/0" for the first device
  -------------------------------------------------------*/

result = tdrv005DevCreate(   "/tdrv005/0",
                             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

Error codes are only set by system functions. The error codes are stored in *errno* and can be read with the function *errnoGet()*.

SEE ALSO

VxWorks Programmer’s Guide: I/O System

3.3 tdrv005PciInit()

NAME

tdrv005PciInit() – Generic PCI device initialization

SYNOPSIS

void tdrv005PciInit()

DESCRIPTION

This function is required only for Intel x86 VxWorks platforms. The purpose is to setup the MMU mapping for all required PCI spaces (base address register) and to enable the TDRV005 device for access.

The global variable *tdrv005Status* 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 tdrv005Status is equal to the number of mapped PCI spaces
0	No TDRV005 device found
< 0	Initialization failed. The value of (tdrv005Status & 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 tdrv005PciInit();

…

tdrv005PciInit();

…
```

4 TDRV005 API Documentation

This TDRV005 Device Driver Application Programming Interface (API) enhances the compatibility of a TDRV005 based application to different operating systems. The API itself uses an abstraction layer called Common Operating System Interface (TCOSI), which hides the different operating system entry functions like open(), close(), read(), write() and ioctl() under a well-defined interface. This results in an operating system independent design of the API.

The TDRV005 API concept helps to provide applications on different operating systems and platforms with only a few changes to the application itself.

4.1 General Functions

4.1.1 tdrv005open()

Name

tdrv005open() – opens a device.

Synopsis

```
int tdrv005open
(
     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.

Return value

If the function succeeds, the return value is an open handle called file descriptor to the specified device. If the function fails, a negative error code is returned.

Errors

TERR_INVALID_HANDLE_VALUE	The specified device does not exist.

Example

```
#include “tdrv005api.h”
TEWS_HANDLE FileDescriptor;

/*
** open file descriptor to device
*/
FileDescriptor = tdrv005open( “/tdrv005/0” );
if (FileDescriptor < 0)
{
          /* handle open error */
}
```

4.1.2 tdrv005close()

Name

tdrv005close() – closes a device.

Synopsis

```
int tdrv005close
(
     TEWS_HANDLE  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.

Return value

TEWS_OK if the device was closed successfully, otherwise a negative error code.

Errors

TERR_INVALID_HANDLE_VALUE	Invalid file descriptor specified.

Example

```
#include “tdrv005api.h”
TEWS_HANDLE   FileDescriptor;
int           result;

/*
** close file descriptor to device
*/
result = tdrv005close( FileDescriptor );
if (result < 0)
{
          /* handle close error */
}
```

4.2 SSI Functions

4.2.1 tdrv005ssiSetup()

Name

tdrv005ssiSetup() – sets up a channel for SSI operation.

Synopsis

```
int tdrv005ssiSetup
(
     TEWS_HANDLE        FileDescriptor,
     unsigned char      Channel,
     TDRV005_SSI_SETUP  *Options
);
```

Description

This function sets up a specific channel to the provided SSI configuration. The function returns immediately to the caller after setting up the corresponding channel.

The SSI channel is not enabled by this command. This must be done by a subsequent call to td005globalChannelEnable (see chapter 4.6.1).

Parameters

FileDescriptor

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

Channel

This value specifies the channel which should be affected. The pre-defined values (TDRV005_CH0 – TDRV005_CH5) must be used. Only one channel may be specified.

Options

This value specifies the necessary configuration options in the structure *TDRV005_SSI_SETUP* with the following layout:

```
typedef struct
{
    TDRV005_SSI_MODE    Mode;
    unsigned char       NumberOfDataBits;
    TDRV005_SSI_CODING  Coding;
    unsigned char       ZeroBits;
    TDRV005_SSI_PARITY  Parity;
    unsigned char       ClockRate;
} TDRV005_SSI_SETUP;
```

Members

Mode

This value specifies the desired operation mode of the SSI interface. Possible values are:

Value	Description
TDRV005_MODE_STANDARD	Standard SSI operation
TDRV005_MODE_LISTENONLY	SSI interface operates in listen-only mode.

NumberOfDataBits

This value specifies the number of data bits to use. Possible values are between 1 and 32.

Coding

This value specifies the desired coding format. Possible values are:

Value	Description
TDRV005_CODING_BINARY	Binary coding is used.
TDRV005_CODING_GRAY	Gray coding is used, data is converted into binary.

ZeroBits

This value specifies the number of zero bits to use in combination with parity. Possible values are either 0 or 1.

Parity

This value specifies what kind of parity bit should be used. Possible values are:

Value	Description
TDRV005_PARITY_NONE	No parity bit is used.
TDRV005_PARITY_EVEN	An even parity bit is used.
TDRV005_PARITY_ODD	An odd parity bit is used.

ClockRate

This value specifies the clock rate for the encoder's serial clock speed. The clock can be programmed in steps of 1µs in the range of 1 to 15.

Return value

TEWS_OK if the channel was configured successfully, otherwise a negative error code.

Errors

TERR_INVALID_CHANNEL	Invalid channel specified.
TERR_INVALID_PARAMETER	Invalid parameter specified, or data buffer null.

Example

```
#include “tdrv005api.h”
TEWS_HANDLE        FileDescriptor;
int                result;
TDRV005_SSI_SETUP  Options;

/*
** setup the counter with appropriate options
*/
Options.Mode                = TDRV005_MODE_STANDARD;
Options.NumberOfDataBits    = 32;
Options.Coding              = TDRV005_CODING_BINARY;
Options.ZeroBits            = 1;
Options.Parity              = TDRV005_PARITY_NONE;
Options.ClockRate           = 10;

result = tdrv005ssiSetup( FileDescriptor, TDRV005_CH0, &Options );
if (result < 0)
{
     /* handle configuration error */
}
```

4.2.2 tdrv005ssiRead()

Name

tdrv005ssiRead() – reads the value of an SSI channel.

Synopsis

```
int tdrv005ssiRead
(
     TEWS_HANDLE   FileDescriptor,
     unsigned char Channel,
     int           Timeout,
     unsigned long *Data,
     unsigned long *Status
);
```

Description

This function reads the value of the corresponding channel's data register. Only the number of previously configured data bits is valid. The function returns to the caller after the desired channel is read or the specified timeout occurred.

Parameters

FileDescriptor

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

Channel

This value specifies the channel which should be affected. The pre-defined values (TDRV005_CH0 – TDRV005_CH5) must be used. Only one channel may be specified.

Timeout

This value specifies the timeout in milliseconds. If the function should wait indefinitely for the data to be valid, *TDRV005_WAIT_FOREVER* must be specified.

Data

This parameter points to an unsigned long value were the data register content is stored.

Status

This parameter points to an unsigned long value were the status register content is stored.

Return value

TEWS_OK if the read operation was successfully, otherwise a negative error code.

Errors

TERR_INVALID_CHANNEL	Invalid channel specified.
TERR_BUSY	Channel is not configured to SSI mode, or there is another job in progress.
TERR_INVALID_PARAMETER	Invalid parameter specified. Data pointer is null.

Example

```
#include “tdrv005api.h”
TEWS_HANDLE   FileDescriptor;
int           result;
unsigned long ssiValue;
unsigned long ssiStatus;

/*
** read the current counter value
*/
result = tdrv005ssiRead( FileDescriptor,
                         TDRV005_CH0,
                         TDRV005_WAIT_FOREVER,
                         &ssiValue,
                         &ssiStatus );
if (result == TEWS_OK)
{
     printf( SSI Value  = 0x%08lX\n”, ssiValue );
     printf( SSI Status = 0x%08lX\n”, ssiStatus );
}
```

4.3 Counter Functions

4.3.1 tdrv005counterSetup()

Name

tdrv005counterSetup() – sets up a channel for counter operation.

Synopsis

```
int tdrv005counterSetup
(
     TEWS_HANDLE             FileDescriptor,
     unsigned char           Channel,
     TDRV005_COUNTER_SETUP   *Options
);
```

Description

This function sets up a specific channel to the provided counter configuration. The function returns immediately to the caller after setting up the corresponding channel.

The counter channel is not enabled by this command. This must be done by a subsequent call to td005globalChannelEnable (see chapter 4.6.1).

Parameters

FileDescriptor

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

Channel

This value specifies the channel which should be affected. The pre-defined values (TDRV005_CH0 – TDRV005_CH5) must be used. Only one channel may be specified.

Options

This value specifies the necessary configuration options in the structure *TDRV005_COUNTER_SETUP* with the following layout:

```
typedef struct
{
    unsigned char           Polarity;
    TDRV005_CNT_INPUT       InputMode;
    TDRV005_CNT_INDEX       IndexControlMode;
    TDRV005_CNT_SCM         SpecialCountMode;
    TDRV005_CNT_CLKDIV      ClockPrescaler;
} TDRV005_COUNTER_SETUP;
```

Members

Polarity

This value specifies the input polarity of the specified channel. The Input Polarity Control can be used to adapt the input to the input source polarity of A, B and I. Use the following predefined values to generate an OR'ed polarity value.

Value	Description
TDRV005_POLARITY_A_LOW	low-active signal, default is high-active
TDRV005_POLARITY_B_LOW	low-active signal, default is high-active
TDRV005_POLARITY_I_LOW	low-active signal, default is high-active

InputMode

The Input Mode determines the input source and how the counter interprets these input signals. Possible values are:

Value	Description	Input Source
TDRV005_INPUT_TIMER_UP	Timer mode Up	internal clock prescaler
TDRV005_INPUT_TIMER_DOWN	Timer mode Down	internal clock prescaler
TDRV005_INPUT_DIRECTION	Direction count	Input A & Input B
TDRV005_INPUT_UPDOWN	Up/Down count	Input A & Input B
TDRV005_INPUT_QUADRATURE_1X	Quadrature count 1x	Input A & Input B
TDRV005_INPUT_QUADRATURE_2X	Quadrature count 2x	Input A & Input B
TDRV005_INPUT_QUADRATURE_4X	Quadrature count 4x	Input A & Input B

IndexControlMode

The Index Control Mode determines how the counter interprets events on the I-input. Possible values are:

Value	Description
TDRV005_ICM_NO_INDEX_CONTROL	no I-control
TDRV005_ICM_LOAD_ON_INDEX	load on index signal
TDRV005_ICM_LATCH_ON_INDEX	latch on index signal
TDRV005_ICM_GATE_ON_INDEX	gate on index signal
TDRV005_ICM_RESET_ON_INDEX	reset on index signal
TDRV005_ICM_REFERENCE_MODE	reference mode (quadrature input mode only)
TDRV005_ICM_AUTO_REFERENCE_MODE	auto-reference mode (quadrature input mode only)
TDRV005_ICM_INDEX_MODE	index mode (quadrature input mode only)

SpecialCountMode

This value specifies the desired special count mode. Possible values are:

Value	Description
TDRV005_SCM_CYCLING_COUNTER	No special count mode, cycling counter.
TDRV005_SCM_DIVIDE_BY_N	Divide-by-N mode.
TDRV005_SCM_SINGLE_CYCLE	Single cycle mode.

ClockPrescaler

This value specifies the internal clock prescaler to be used. Possible values are:

Value	Description
TDRV005_CLKDIV_1X	Prescaler 1x, 32 MHz clock
TDRV005_CLKDIV_2X	Prescaler 2x, 16 MHz clock
TDRV005_CLKDIV_4X	Prescaler 4x, 8 MHz clock
TDRV005_CLKDIV_8X	Prescaler 8x, 4 MHz clock

Return value

TEWS_OK if the counter was configured successfully, otherwise a negative error code.

Errors

TERR_INVALID_CHANNEL	Invalid channel specified.
TERR_INVALID_PARAMETER	Invalid parameter specified.

Example

```
#include “tdrv005api.h”
TEWS_HANDLE             FileDescriptor;
int                     result;
TDRV005_COUNTER_SETUP   Options;

/*
** setup the counter with appropriate options
*/
Options.Polarity            = TDRV005_POLARITY_A_LOW |
                              TDRV005_POLARITY_B_LOW |
                              TDRV005_POLARITY_I_LOW;
Options.InputMode           = TDRV005_INPUT_UPDOWN;
Options.IndexControlMode    = TDRV005_ICM_NO_INDEX_CONTROL;
Options.SpecialCountMode    = TDRV005_SCM_CYCLING_COUNTER;
Options.ClockPrescaler      = TDRV005_CLKDIV_8X;

result = tdrv005counterSetup( FileDescriptor, TDRV005_CH0, &Options );
if (result < 0)
{
        /* handle configuration error */
}
```

4.3.2 tdrv005counterRead()

Name

tdrv005counterRead() – reads the value of a counter channel.

Synopsis

```
int tdrv005counterRead
(
     TEWS_HANDLE   FileDescriptor,
     unsigned char Channel,
     unsigned long *Data,
     unsigned long *Status
);
```

Description

This function reads the value of the corresponding channel's data register. The function returns immediately to the caller.

Parameters

FileDescriptor

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

Channel

This value specifies the channel on which the specified event should occur. The pre-defined values (TDRV005_CH0 – TDRV005_CH5) must be used. Only one channel may be specified.

Data

This parameter points to an unsigned long value were the data register content is stored.

Status

This parameter points to an unsigned long value were the status register content is stored.

Return value

TEWS_OK if the read operation was successfully, otherwise a negative error code.

Errors

TERR_INVALID_CHANNEL	Invalid channel specified.
TERR_INVALID_PARAMETER	Invalid parameter specified. Data pointer is null.
TERR_BUSY	Channel is not configured to counter mode.

Example

```
#include “tdrv005api.h”
TEWS_HANDLE   FileDescriptor;
int           result;
unsigned long CounterValue, Status;

/*
** read the current counter value
*/
result = tdrv005counterRead( FileDescriptor,
                             TDRV005_CH0,
                             &CounterValue,
                             &Status );
if (result == TEWS_OK)
{
     printf( Counter Value = 0x%08lX\n”, CounterValue );
     printf( Status  Value = 0x%08lX\n”, Status );
}
```

4.3.3 tdrv005counterPreloadSet()

Name

tdrv005counterPreloadSet() – sets the preload register of a counter channel.

Synopsis

```
int tdrv005counterPreloadSet
(
     TEWS_HANDLE   FileDescriptor,
     unsigned char Channel,
     unsigned long PreloadValue
);
```

Description

Set the counter's preload register to the supplied value. The function returns immediately to the caller.

Parameters

FileDescriptor

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

Channel

This value specifies the channel which should be affected. The pre-defined values (TDRV005_CH0 – TDRV005_CH5) must be used. Only one channel may be specified.

PreloadValue

This value specifies the new value of the channel's preload register.

Return value

TEWS_OK if the counter preload register was set successfully, otherwise a negative error code.

Errors

TERR_INVALID_CHANNEL	Invalid channel specified.
TERR_INVALID_PARAMETER	Invalid parameter, data buffer is null.
TERR_BUSY	Channel is not configured to counter mode.

Example

```
#include “tdrv005api.h”
TEWS_HANDLE   FileDescriptor;
int           result;
unsigned long PreloadValue;

/*
** load counter value
*/
PreloadValue = 0x12345678;
result = tdrv005counterPreloadSet( TDRV005_CH0, PreloadValue );
if (result < 0)
{
         /* handle error */
}
```

4.3.4 tdrv005counterLoad()

Name

tdrv005counterLoad() – loads the preload register into the counter channel.

Synopsis

```
int tdrv005counterLoad
(
     TEWS_HANDLE   FileDescriptor,
     unsigned char Channel
);
```

Description

Load the counter to the previously specified value of the counter preload register. The function returns immediately to the caller. To simultaneously load multiple channels, please refer to chapter 4.6.4.

Parameters

FileDescriptor

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

Channel

This value specifies the channel which should be affected. The pre-defined values (TDRV005_CH0 – TDRV005_CH5) must be used. Only one channel may be specified.

Return value

TEWS_OK if the counter was loaded successfully, otherwise a negative error code.

Errors

TERR_INVALID_CHANNEL	Invalid channel specified.
TERR_INVALID_PARAMETER	Invalid parameter, data buffer is null.
TERR_BUSY	Channel is not configured to counter mode.

Example

```
#include “tdrv005api.h”
TEWS_HANDLE   FileDescriptor;
int           result;

/*
** load counter value
*/
result = tdrv005counterLoad( FileDescriptor, TDRV005_CH0 );
if (result < 0)
{
          /* handle error */
}
```

4.3.5 tdrv005counterReset()

Name

tdrv005counterReset() – resets the counter channel.

Synopsis

```
int tdrv005counterReset
(
     TEWS_HANDLE   FileDescriptor,
     unsigned char Channel
);
```

Description

Reset the counter value of the specified channel. The function returns immediately to the caller.

Parameters

FileDescriptor

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

Channel

This value specifies the channel which should be affected. The pre-defined values (TDRV005_CH0 – TDRV005_CH5) must be used. Only one channel may be specified.

Return value

TEWS_OK if the counter was reset successfully, otherwise a negative error code.

Errors

TERR_INVALID_CHANNEL	Invalid channel specified.
TERR_INVALID_PARAMETER	Invalid parameter, data buffer is null.
TERR_BUSY	Channel is not configured to counter mode.

Example

```
#include “tdrv005api.h”
TEWS_HANDLE   FileDescriptor;
int           result;

/*
** reset counter value
*/
result = tdrv005counterReset( FileDescriptor, TDRV005_CH0 );
if (result < 0)
{
          /* handle error */
}
```

4.3.6 tdrv005counterWaitMatch()

Name

tdrv005counterWaitMatch() – waits for a counter-match event.

Synopsis

```
int tdrv005counterWaitMatch
(
     TEWS_HANDLE   FileDescriptor,
     unsigned char Channel,
     unsigned long CompareValue,
     int           Timeout
);
```

Description

Waits until the counter value matches the provided counter compare value. The function returns to the caller if the counter matches the provided compare value, or the specified timeout occurred.

Parameters

FileDescriptor

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

Channel

This value specifies the channel on which the specified event should occur. The pre-defined values (TDRV005_CH0 – TDRV005_CH5) must be used. Only one channel may be specified.

CompareValue

This parameter specifies the value to which the counter should be compared.

Timeout

This value specifies the timeout in milliseconds. If the function should wait indefinitely for the event to occur, TDRV005_WAIT_FOREVER must be specified.

Return value

TEWS_OK if the counter event occurred successfully, otherwise a negative error code.

Errors

TERR_INVALID_CHANNEL	Invalid channel specified.
TERR_INVALID_PARAMETER	Invalid parameter, data buffer is null.
TERR_TIMEOUT	The event has not occurred, timeout.
TERR_BUSY	Channel is not configured to counter mode, or a counter-match job is already pending.

Example

```
#include “tdrv005api.h”
TEWS_HANDLE   FileDescriptor;
int           result;
unsigned long MatchValue;

/*
** wait indefinitely for counter match event
*/
MatchValue = 0x12345678;
result = tdrv005counterWaitMatch( FileDescriptor,
                                  TDRV005_CH0,
                                  MatchValue,
                                  TDRV005_WAIT_FOREVER );
if (result < 0)
{
         /* handle error */
}
```

4.3.7 tdrv005counterWaitControlModeEvent()

Name

tdrv005counterWaitControlModeEvent() – waits for a counter-control-mode event.

Synopsis

```
int tdrv005counterWaitControlModeEvent
(
     TEWS_HANDLE   FileDescriptor,
     unsigned char Channel,
     int           Timeout
);
```

Description

Wait for the control mode event of the counter. The function returns to the caller if the configured control mode event or the specified timeout occurred.

Parameters

FileDescriptor

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

Channel

This value specifies the channel on which the specified event should occur. The pre-defined values (TDRV005_CH0 – TDRV005_CH5) must be used. Only one channel may be specified.

Timeout

This value specifies the timeout in milliseconds. If the function should wait indefinitely for the event to occur, TDRV005_WAIT_FOREVER must be specified.

Return value

TEWS_OK if the counter event occurred successfully, otherwise a negative error code.

Errors

TERR_INVALID_CHANNEL	Invalid channel specified.
TERR_INVALID_PARAMETER	Invalid parameter, data buffer is null.
TERR_TIMEOUT	The event has not occurred, timeout.
TERR_BUSY	Channel is not configured to counter mode, or a counter-control job is already pending.

Example

```
#include “tdrv005api.h”
TEWS_HANDLE   FileDescriptor;
int           result;

/*
** wait indefinitely for counter control mode event
*/
result = tdrv005counterWaitControlModeEvent( FileDescriptor,
                                             TDRV005_CH0,
                                             TDRV005_WAIT_FOREVER );
if (result < 0)
{
         /* handle error */
}
```

4.4 Timer Functions

4.4.1 tdrv005timerSetup()

Name

tdrv005timerSetup() – sets up the timer.

Synopsis

```
int tdrv005timerSetup
(
     TEWS_HANDLE        FileDescriptor,
     TDRV005_CNT_CLKFRQ      ClockFrequency,
     unsigned short     PreloadValue
);
```

Description

This function sets up the onboard timer to the provided configuration. The function returns immediately to the caller. The interval timer remains stopped after a call to this function.

Parameters

FileDescriptor

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

ClockFrequency

This value specifies the clock frequency used as clock source for the timer. Possible values are:

Value	Description
TDRV005_CLKFRQ_1MHZ	1 MHz clock
TDRV005_CLKFRQ_2MHZ	2 MHz clock
TDRV005_CLKFRQ_4MHZ	4 MHz clock
TDRV005_CLKFRQ_8MHZ	8 MHz clock

PreloadValue

This value specifies the preload value of the timer. If the interval timer is running, this value is loaded automatically every time the timer expires. The preload value is of 16bit width.

Return value

TEWS_OK if the timer was configured successfully, otherwise a negative error code.

Errors

TERR_INVALID_PARAMETER	Invalid parameter specified.

Example

```
#include “tdrv005api.h”
TEWS_HANDLE   FileDescriptor;
int           result;

/*
** setup the interval timer with an interrupt frequency of 100 Hz
*/
result = tdrv005timerSetup( FileDescriptor,
                            TDRV005_CLKFRQ_1MHZ,
                            10000 );
if (result < 0)
{
         /* handle error */
}
```

4.4.2 tdrv005timerStart()

Name

tdrv005timerStart() – starts the timer.

Synopsis

```
int tdrv005timerStart
(
     TEWS_HANDLE FileDescriptor
);
```

Description

Start the previously configured timer. The function returns immediately to the caller.

Parameters

FileDescriptor

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

Return value

TEWS_OK if the timer was started successfully, otherwise a negative error code.

Errors

TERR_NO_CONFIGURATION	The timer was not configured properly.

Example

```
#include “tdrv005api.h”
TEWS_HANDLE   FileDescriptor;
int           result;

/*
** start the previously configured interval timer
*/
result = tdrv005timerStart( FileDescriptor );
if (result < 0)
{
          /* handle error */
}
```

4.4.3 tdrv005timerStop()

Name

tdrv005timerStop() – stops the timer.

Synopsis

```
int tdrv005timerStop
(
     TEWS_HANDLE FileDescriptor
);
```

Description

Stop the previously configured timer. The function returns immediately to the caller.

Parameters

FileDescriptor

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

Return value

TEWS_OK if the timer was stopped successfully, otherwise a negative error code.

Errors

TERR_NO_CONFIGURATION	The timer was not configured properly.

Example

```
#include “tdrv005api.h”
TEWS_HANDLE   FileDescriptor;
int           result;

/*
** stop the interval timer
*/
result = tdrv005timerStop( FileDescriptor );
if (result < 0)
{
          /* handle error */
}
```

4.4.4 tdrv005timerRead()

Name

td005timerRead() – reads the current timer value.

Synopsis

```
int tdrv005timerRead(
     TEWS_HANDLE        FileDescriptor,
     unsigned short     *Data
);
```

Description

Read the current value of the timer. The function returns immediately to the caller.

Parameters

FileDescriptor

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

Data

This parameter points to an unsigned short value where the data register content is stored.

Return value

TEWS_OK if the read operation was successfully, otherwise a negative error code.

Errors

TERR_INVALID_PARAMETER	Invalid parameter specified, data pointer is null.

Example

```
#include “tdrv005api.h”
TEWS_HANDLE        FileDescriptor;
int                result;
unsigned short     TimerValue;

/*
** read the current interval timer value
*/
result = tdrv005timerRead( FileDescriptor, &TimerValue );
if (result == TEWS_OK)
{
     printf( Timer Value = 0x%04X\n”, TimerValue );
}
```

4.4.5 tdrv005timerWait()

Name

td005timerWait() – waits for a timer event.

Synopsis

```
int tdrv005timerWait
(
     TEWS_HANDLE    FileDescriptor,
     int            Timeout
);
```

Description

Wait for the timer event or the specified timeout to occur. The function returns to the caller if the timer has expired, or the specified timeout occurred.

Parameters

FileDescriptor

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

Timeout

This value specifies the timeout in milliseconds. If the function should wait indefinitely for the event to occur, TDRV005_WAIT_FOREVER must be specified.

Return value

TEWS_OK if the timer event occurred successfully, otherwise a negative error code.

Errors

TERR_NOT_RUNNING	The timer is not running.
TERR_NO_CONFIGURATION	The timer was not configured properly.
TERR_TIMEOUT	The timer event has not occurred, timeout.

Example

```
#include “tdrv005api.h”
TEWS_HANDLE   FileDescriptor;
int           result;

/*
** wait indefinitely for an interval timer event
*/
result = tdrv005timerWait( FileDescriptor, TDRV005_WAIT_FOREVER );
if (result < 0)
{
          /* handle error */
}
```

4.4.6 tdrv005timerMultipleChannelReadSetup()

Name

tdrv005timerMultipleChannelReadSetupt() – sets up channels for multiple read.

Synopsis

```
int tdrv005timerMultipleChannelReadSetup
(
     TEWS_HANDLE    FileDescriptor,
     unsigned char  Channel
);
```

Description

Configure specified channels for simultaneous sampling triggered by the timer. The function returns immediately to the caller.

Channels configured to SSI listen-only mode may not be used with this function.

Parameters

FileDescriptor

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

Channel

This value specifies the channels which should be read simultaneously. The pre-defined values (TDRV005_CH0 – TDRV005_CH5) must be used. Multiple channels may be OR'ed to one value.

Return value

TEWS_OK if the multiple-channel-read operation was configured successfully, otherwise a negative error code.

Errors

TERR_INVALID_CHANNEL	Invalid channel specified, or a specified channel is not configured properly.
TERR_INVALID_PARAMETER	Invalid parameter, buffer is NULL.
TERR_BUSY	Specified channel is busy with an SSI job.

Example

```
#include “tdrv005api.h”
TEWS_HANDLE   FileDescriptor;
int           result;

/*
** setup channels 0+5 for simultaneous sampling triggered by timer
*/
result = tdrv005timerMultipleChannelReadSetup( FileDescriptor,
                                                TDRV005_CH0 | TDRV005_CH5 );
if (result < 0)
{
         /* handle error */
}
```

4.4.7 tdrv005timerMultipleChannelReadWait()

Name

tdrv005timerMultipleChannelReadWait() – waits for the simultaneous data timer event.

Synopsis

```
int tdrv005timerMultipleChannelReadWait
(
     TEWS_HANDLE             FileDescriptor,
     TDRV005_MULTIPLEVALUES  *MultipleValues,
     unsigned long           &Timestamp,
     int                     *MoreDataAvailable,
     int                     Timeout
);
```

Description

Return the values of simultaneously sampled channels. An array to store all channel values must be supplied to this function. The function waits for the timer event to occur on which the previously configured channels are sampled, or the specified timeout occurred. Before using this function the timer must be configured properly.

Parameters

FileDescriptor

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

MultipleValues

This is a pointer to a TDRV005_MULTIPLEVALUES structure, where the read values are stored. Note that only the previously configured channels return valid data, other data entries must be ignored. The TDRV005_MULTIPLEVALUES structure has the following layout:

```
typedef struct
{
      TDRV005_VALUE_BUF  Channel[6];
} TDRV005_MULTIPLEVALUES;
```

Members

Channel

This parameter is an array of a TDRV005_VALUE_BUF structure which holds the returned channel data. The value for channel 0 is returned at array index 0, channel 5's value is located at array index 5. The TDRV005_VALUE_BUF structure has the following layout:

```
typedef struct
{
        unsigned long       Value;
        unsigned long       Status;
} TDRV005_VALUE_BUF;
```

Members

Value

This parameter holds the returned channel value.

Status

This parameter holds the returned channel status.

Timestamp

This is a pointer to an unsigned long value where the number of occurred timer interrupts is returned.

MoreDataAvailable

This is a pointer to a boolean integer value. The value is TRUE if additional data is available. This might happen if the timer event triggering the multiple-channel-read operation appears too fast. An additional call to td005timerMultipleChannelReadWait must be performed.

Timeout

This value specifies the timeout in milliseconds. If the function should wait indefinitely for the event to occur, TDRV005_WAIT_FOREVER must be specified.

Return value

TEWS_OK if the timer event triggering the multiple-channel-read operation occurred successfully and data is available, otherwise a negative error code.

Errors

TERR_NOT_RUNNING	The timer is not running.
TERR_NO_CONFIGURATION	The timer was not configured properly.
TERR_INVALID_PARAMETER	Invalid parameter, output buffer is NULL.
TERR_TIMEOUT	The timer event has not occurred, timeout.
TERR_BUSY	Another job is already pending.

Example

```
#include “tdrv005api.h”
TEWS_HANDLE             FileDescriptor;
int                     result;
int                     MoreDataAvailable;
unsigned long           Timestamp;
TDRV005_MULTIPLEVALUES  MultipleValues;

/*
** read channels triggered by timer
*/
result = tdrv005timerMultipleChannelReadWait( FileDescriptor,
                                              &MultipleValues,
                                              &Timestamp,
                                              &MoreDataAvailable,
                                              TDRV005_WAIT_FOREVER );
if (result == TEWS_OK)
{
     printf( “Timestamp  = %ld\n”, Timestamp );
     printf( “Channel(0) = 0x%08lX\n”,
                    MultipleValues.Channel[0].Value );
     printf( “Channel(5) = 0x%08lX\n”,
                    MultipleValues.Channel[5].Value );
} else {
          /* handle error */
}
```

4.5 Digital Input Functions

4.5.1 tdrv005digitalRead()

Name

tdrv005digitalRead() – reads the digital input lines.

Synopsis

```
int tdrv005digitalRead
(
     TEWS_HANDLE   FileDescriptor,
     unsigned char *Data
);
```

Description

Reads the current values of all digital input lines. The function returns immediately to the caller.

Parameters

FileDescriptor

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

Data

This parameter points to an *unsigned char* value were the data register content is stored. Channel 0 is represented by bit 0, channel 5 is represented by bit 5 of the returned byte value.

Return value

TEWS_OK if the read operation was successfully, otherwise a negative error code.

Errors

TERR_INVALID_PARAMETER	Invalid parameter specified, data pointer is null.

Example

```
#include “tdrv005api.h”
TEWS_HANDLE   FileDescriptor;
int           result;
unsigned char DigitalValue;

/*
** read the current interval timer value
*/
result = tdrv005digitalRead( FileDescriptor, &DigitalValue);
if (result == TEWS_OK)
{
     printf( “Digital Value = 0x%02X\n”, DigitalValue );
}
```

4.5.2 tdrv005digitalWait()

Name

td005digitalWait() – waits for an event on a digital input line.

Synopsis

```
int tdrv005digitalWait
(
     TEWS_HANDLE        FileDescriptor,
     unsigned char      Channel,
     TDRV005_TRANSITION Transition,
     int                Timeout
);
```

Description

Wait for the specified transition (rising or falling edge) on the specified digital input line. The function returns to the caller if the specified transition or the specified timeout occurred.

Parameters

FileDescriptor

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

Channel

This value specifies the channel on which the specified event should occur. The pre-defined values (TDRV005_CH0 – TDRV005_CH5) must be used. Only one channel may be specified.

Transition

This value specifies the transition on which the specified event should occur.

Value	Description
TDRV005_TR_RISING_EDGE	Rising edge on digital input
TDRV005_TR_FALLING_EDGE	Falling edge on digital input

Timeout

This value specifies the timeout in milliseconds. If the function should wait indefinitely for the event to occur, TDRV005_WAIT_FOREVER must be specified.

Return value

TEWS_OK if the timer event occurred successfully, otherwise a negative error code.

Errors

TERR_INVALID_CHANNEL	Invalid channel specified.
TERR_INVALID_PARAMETER	Invalid parameter specified.
TERR_TIMEOUT	The event has not occurred, timeout.

Example

```
#include “tdrv005api.h”
TEWS_HANDLE   FileDescriptor;
int           result;

/*
** wait indefinitely for a rising edge on digital input of channel 1
*/
result = tdrv005digitalWait( FileDescriptor,
                             TDRV005_CH1,
                             TDRV005_TR_RISING_EDGE,
                             TDRV005_WAIT_FOREVER );
if (result < 0)
{
     /* handle error */
}
```

4.6 Global Operation Functions

4.6.1 tdrv005globalChannelEnable()

Name

tdrv005globalChannelEnable() – globally enables multiple channels.

Synopsis

```
int tdrv005globalChannelEnable
(
     TEWS_HANDLE   FileDescriptor,
     unsigned char Channels
);
```

Description

Enable multiple channels (SSI or Counter) simultaneously. The function returns immediately to the caller.

Parameters

FileDescriptor

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

Channels

This value specifies the channels which should be enabled. The pre-defined values (TDRV005_CH0 – TDRV005_CH5) must be used. Multiple channels may be OR'ed to one value.

Return value

TEWS_OK if the specified channels were enabled successfully, otherwise a negative error code.

Errors

TERR_INVALID_CHANNEL	Invalid channel specified.
TERR_INVALID_PARAMETER	Invalid parameter, data buffer is null.

Example

```
#include “tdrv005api.h”
TEWS_HANDLE   FileDescriptor;
int           result;

/*
** enable channel 0 and channel 5
*/
result = tdrv005globalChannelEnable ( FileDescriptor,
                                      TDRV005_CH0 | TDRV005_CH5 );
if (result < 0)
{
          /* handle error */
}
```

4.6.2 tdrv005globalChannelDisable()

Name

tdrv005globalChannelDisable() – globally disables multiple channels.

Synopsis

```
int tdrv005globalChannelDisable
(
     TEWS_HANDLE   FileDescriptor,
     unsigned char Channels
);
```

Description

Disable multiple channels (SSI or Counter) simultaneously. The function returns immediately to the caller.

Parameters

FileDescriptor

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

Channels

This value specifies the channels which should be disabled. The pre-defined values (TDRV005_CH0 – TDRV005_CH5) must be used. Multiple channels may be OR'ed to one value.

Return value

TEWS_OK if the specified channels were disabled successfully, otherwise a negative error code.

Errors

TERR_INVALID_CHANNEL	Invalid channel specified.
TERR_INVALID_PARAMETER	Invalid parameter, data buffer is null.

Example

```
#include “tdrv005api.h”
TEWS_HANDLE   FileDescriptor;
int           result;

/*
** disable channel 0 and channel 5
*/
result = tdrv005globalChannelDisable ( FileDescriptor,
                                       TDRV005_CH0 | TDRV005_CH5 );
if (result < 0)
{
          /* handle error */
}
```

4.6.3 tdrv005globalCounterPreloadSet()

Name

tdrv005globalCounterPreloadSet() – globally sets counter preload registers.

Synopsis

```
int tdrv005globalCounterPreloadSet
(
     TEWS_HANDLE             FileDescriptor,
     unsigned char           Channels,
     TDRV005_MULTIPLEVALUES  *MultipleValues
);
```

Description

Perform a simultaneous setup of the preload registers of specified counter channels. The desired values must be supplied to this function as well as the channel numbers which are affected. The function returns immediately to the caller.

Parameters

FileDescriptor

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

Channels

This value specifies the channels which should be preloaded. The pre-defined values (TDRV005_CH0 – TDRV005_CH5) must be used. Multiple channels may be OR'ed to one value.

MultipleValues

This is a pointer to a TDRV005_MULTIPLEVALUES structure, where the read values are stored. The TDRV005_MULTIPLEVALUES structure has the following layout:

```
typedef struct
{
      TDRV005_VALUE_BUF  Channel[6];
} TDRV005_MULTIPLEVALUES;
```

Members

Channel

This parameter is an array of a TDRV005_VALUE_BUF structure which holds the preload data. The value for channel 0 is located at array index 0, channel 5's value is located at array index 5. The TDRV005_VALUE_BUF structure has the following layout:

```
typedef struct
{
        unsigned long       Value;
        unsigned long       Status;
} TDRV005_VALUE_BUF;
```

Members

Value

This parameter holds the preload channel value.

Status

This parameter is not used for this function.

Return value

TEWS_OK if the preload registers were set successfully, otherwise a negative error code.

Errors

TERR_INVALID_CHANNEL	Invalid channel specified.
TERR_INVALID_PARAMETER	Invalid parameter, data pointer is null.
TERR_BUSY	Channel is not configured to counter mode.

Example

```
#include “tdrv005api.h”
TEWS_HANDLE             FileDescriptor;
int                     result;
TDRV005_MULTIPLEVALUES  Values;

/*
** preload channel 0 and channel 5
*/
Values[0].Value = 0x00000000;
Values[5].Value = 0x50000000;
result = tdrv005globalCounterPreloadSet ( FileDescriptor,
                                          TDRV005_CH0 | TDRV005_CH5,
                                          Values );
if (result < 0)
{
         /* handle error */
}
```

4.6.4 tdrv005globalCounterLoad()

Name

tdrv005globalCounterLoad() – globally loads preload registers into counters.

Synopsis

```
int tdrv005globalCounterLoad
(
     TEWS_HANDLE   FileDescriptor,
     unsigned char Channels
);
```

Description

Perform a simultaneous preload of specified counter channels. The values stored in the corresponding preload registers are loaded into the specified counters simultaneously. The function returns immediately to the caller.

Parameters

FileDescriptor

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

Channels

This value specifies the channels which should be preloaded. The pre-defined values (TDRV005_CH0 – TDRV005_CH5) must be used. Multiple channels may be OR'ed to one value.

Return value

TEWS_OK if the specified counters were loaded successfully, otherwise a negative error code.

Errors

TERR_INVALID_CHANNEL	Invalid channel specified.
TERR_INVALID_PARAMETER	Invalid parameter, data buffer is null.
TERR_BUSY	Channel is not configured to counter mode.

Example

```
#include “tdrv005api.h”
TEWS_HANDLE   FileDescriptor;
int           result;

/*
** load channel 0 and channel 5
*/
result = tdrv005globalCounterLoad ( FileDescriptor,
                                    TDRV005_CH0 | TDRV005_CH5 );
if (result < 0)
{
         /* handle error */
}
```

4.6.5 tdrv005globalMultipleChannelRead()

Name

tdrv005globalMultipleChannelRead() – reads multiple channels simultaneously.

Synopsis

```
int tdrv005globalMultipleChannelRead
(
     TEWS_HANDLE             FileDescriptor,
     unsigned char           Channels,
     TDRV005_MULTIPLEVALUES  *MultipleValues
);
```

Description

Return the values of simultaneously sampled channels. The desired channel numbers must be supplied to this function as well as an array to store all channel values. The function returns to the caller after the read operation is finished.

Channels configured to SSI listen-only mode may not be used with this function.

Parameters

FileDescriptor

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

Channels

This value specifies the channels which should be read simultaneously. The pre-defined values (TDRV005_CH0 – TDRV005_CH5) must be used. Multiple channels may be OR'ed to one value.

MultipleValues

This is a pointer to a TDRV005_MULTIPLEVALUES structure, where the read values are stored. The returned values are only valid for channels enabled by the parameter *Channels*. The TDRV005_MULTIPLEVALUES structure has the following layout:

```
typedef struct
{
      TDRV005_VALUE_BUF  Channel[6];
} TDRV005_MULTIPLEVALUES;
```

Members

Channel

This parameter is an array of a TDRV005_VALUE_BUF structure which holds the returned channel data. The value for channel 0 is returned at array index 0, channel 5's value is located at array index 5. The TDRV005_VALUE_BUF structure has the following layout:

```
typedef struct
{
        unsigned long       Value;
        unsigned long       Status;
} TDRV005_VALUE_BUF;
```

Members

Value

This parameter holds the returned channel value.

Status

This parameter holds the returned channel status.

Return value

TEWS_OK if the channel data is read successfully, otherwise a negative error code.

Errors

TERR_INVALID_CHANNEL	Invalid channel specified.
TERR_INVALID_PARAMETER	Invalid parameter, data pointer is null, or a specified channel is not configured properly.
TERR_BUSY	A MultipleChannelRead job is already pending, or a channel is busy with an SSI job.
TERR_TIMEOUT	Internal timeout. The values couldn't be retrieved within 2 seconds.

Example

```
#include “tdrv005api.h”
TEWS_HANDLE             FileDescriptor;
int                     result;
TDRV005_MULTIPLEVALUES  MultipleValues;

/*
** read channel 0 and channel 5 simultaneously
*/
result = tdrv005globalMultipleChannelRead ( FileDescriptor,
                                            TDRV005_CH0 | TDRV005_CH5,
                                            &MultipleValues );
if (result == TEWS_OK)
{
     printf( “Channel(0) = 0x%08lX\n”, MultipleValues.Channel[0].Value );
     printf( “Channel(5) = 0x%08lX\n”, MultipleValues.Channel[5].Value );
} else {
     /* handle error */
}
```