*The Embedded I/O Company*

TEWS TECHNOLOGIES

# TPMC150-SW-42

## VxWorks Device Driver

4, 3, 2 or 1 Channel Synchro/Resolver-to-Digital Converter

Version 4.2.x

## User Manual

Issue 4.2.0

March 2021

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

## TPMC150-SW-42

VxWorks Device Driver

4, 3, 2 or 1 Channel Synchro/Resolver-to-Digital Converter

Supported Modules:
TPMC150



| Issue | Description | Date |
|---|---|---|
| 1.0 | First Issue | July 3, 2003 |
| 1.0.1 | File list changed | April 1, 2005 |
| 2.0.0 | General revision, VxBus support added, API added | September 8, 2010 |
| 3.0.0 | 64bit support added, argument types changed | December 16, 2011 |
| 4.0.0 | API-functions modified, File-list modified<br>Description “Basic I/O Functions” removed<br>New chapter “Debugging and Diagnostic” | June 25, 2013 |
| 4.1.0 | VxWorks 7 support added<br>New installation guide | September 17, 2019 |
| 4.2.0 | ioctl for RTP-Support modified | March 25, 2021 |

# Table of Contents

# 1 Introduction

The TPMC150-SW-42 VxWorks device driver software allows the operation of the supported PMC conforming to the VxWorks I/O system specification.

The TPMC150-SW-42 release contains independent driver sources for the old legacy (pre-VxBus) and the new VxBus-enabled (GEN1 and GEN2) driver model. The VxBus-enabled driver is recommended for new developments with later VxWorks 6.x release and mandatory for VxWorks 64-bit and SMP systems.

Both drivers, legacy and VxBus, share the same application programming interface (API).

Both drivers invoke a mutual exclusion and binary semaphore mechanism to prevent simultaneous requests by multiple tasks from interfering with each other.

The TPMC150-SW-42 device driver supports the following features:

- read converter data
- configure converter
- configure (synchron) read for multiple converters
- read encoder data
- configure encoder
- configure (synchron) read for multiple encoders
- read current state of digital input lines
- wait for digital input events

The TPMC150-SW-42 supports the modules listed below:

| | | |
|---|---|---|
| TPMC150-10 | 4 Channel Synchro/Resolver-to-Digital Converter | (PMC) |
| TPMC150-11 | 3 Channel Synchro/Resolver-to-Digital Converter | (PMC) |
| TPMC150-12 | 2 Channel Synchro/Resolver-to-Digital Converter | (PMC) |
| TPMC150-13 | 1 Channel Synchro/Resolver-to-Digital Converter | (PMC) |

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

| |
|---|
| TPMC150 User Manual |
| TEWS TECHNOLOGIES VxWorks Device Drivers - Installation Guide |

# 2 API Documentation

## 2.1 General Functions

### 2.1.1 tpmc150Open

#### NAME

tpmc150Open – opens a device.

#### SYNOPSIS

```
TPMC150_HANDLE tpmc150Open
(
      char          *DeviceName
)
```

#### DESCRIPTION

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

#### PARAMETERS

*DeviceName*

This parameter points to a null-terminated string that specifies the name of the device. The first TPMC150 device is named “/tpmc150/0”, the second device is named “/tpmc150/1” and so on.

#### EXAMPLE

```
#include “tpmc150api.h”

TPMC150_HANDLE     hdl;

/*
** open the specified device
*/
hdl = tpmc150Open(“/tpmc150/0”);
if (hdl == NULL)
{
     /* handle open error */
}
```

## RETURNS

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

## ERROR CODES

The error codes are stored in *errno.*

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

## 2.1.2 tpmc150Close

### NAME

tpmc150Close – closes a device.

### SYNOPSIS

```
TPMC150_STATUS tpmc150Close
(
     TPMC150_HANDLE       hdl
)
```

### DESCRIPTION

This function closes previously opened devices.

### PARAMETERS

*hdl*

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

### EXAMPLE

```
#include “tpmc150api.h”

TPMC150_HANDLE     hdl;
TPMC150_STATUS     result;

/*
** close the device
*/
result = tpmc150Close(hdl);
if (result != TPMC150_OK)
{
     /* handle close error */
}
```

## RETURNS

On success, TPMC150_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

| Error Code | Description |
|---|---|
| TPMC150_ERR_INVALID_HANDLE | The specified device handle is invalid |

# 2.2 Device Access Functions

## 2.2.1 tpmc150ReadConverter

### NAME

tpmc150ReadConverter – read value from converter channel

### SYNOPSIS

```
TPMC150_STATUS tpmc150ReadConverter
(
    TPMC150_HANDLE      hdl,
    int                 chanNo,
    unsigned short      *pValue,
    unsigned int        *pState
)
```

### DESCRIPTION

This function reads the value from the specified converter channel. Depending on the configuration the function will return the current value of the converter or a value latched by a read of another converter channel.

### PARAMETERS

*hdl*

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

*chanNo*

This parameter specifies the converter channel. Allowed values are 1 up to the number of available converter channels on the specified device.

*pValue*

This parameter points to a buffer, where the function will store the read converter value.

*pState*

This parameter points to a buffer, where the function will store the status data of the converter. The following flags may be set:

| Flag | Description |
|---|---|
| TPMC150_IO_F_CONDATA_BIT | If this bit is set the Built-in-Self-Test failed |
| TPMC150_IO_F_CONDATA_NOADAMOUNT | If this bit is set there is no adapter mounted |
| TPMC150_IO_F_CONDATA_MOTDIR | This bit indicates the direction of the motion.<br>0: down (clockwise, angle decreasing)<br>1: up (counter clockwise, angle increasing) |

## EXAMPLE

```
#include “tpmc150api.h”

TPMC150_HANDLE     hdl;
TPMC150_STATUS     result;
unsigned short     convVal;
unsigned int       convState;

/* read current value and state of converter channel 3 */
result = tpmc150ReadConverter(hdl,
                              3,
                              &convVal,
                              &convState);
if (result != TPMC150_OK)
{
     /* handle error */
}
else
{
     printf(“converter #3:\n”);
     printf(“    value: 0x%04X\n”, convVal);
     printf(“        Direction: %s\n”,
          (convState & TPMC150_IO_F_CONDATA_MOTDIR) ? “up” : “down”);
     if (convState & TPMC150_IO_F_CONDATA_BIT)
          printf(“        Built_in_Self_Test failed\n”);
     if (convState & TPMC150_IO_F_CONDATA_NOADAMOUNT)
          printf(“        No adapter mounted\n”);
}
```

## RETURN VALUE

On success, TPMC150_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

| Error Code | Description |
|---|---|
| TPMC150_ERR_INVALID_HANDLE | The specified device handle is invalid. |
| TPMC150_ERR_INVAL | Invalid input argument specified or NULL pointer referenced |

### 2.2.2 tpmc150ConfigureConverter

#### NAME

tpmc150ConfigureConverter – configure converter channel

#### SYNOPSIS

```
TPMC150_STATUS tpmc150ConfigureConverter
(
    TPMC150_HANDLE      hdl,
    int                 chanNo,
    int                 resolution,
    unsigned int        flags
)
```

#### DESCRIPTION

This function configures the specified converter channel.

#### PARAMETERS

*hdl*

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

*chanNo*

This parameter specifies the converter channel. Allowed values are 1 up to the number of available converter channels on the specified device.

*resolution*

This argument specifies the resolution in bits. Allowed resolutions are 10, 12, 14 and 16 bit.

*flags*

This argument specifies a set of bit flags for configuration:

| Value | Description |
|---|---|
| TPMC150_LATCH_ENABLE | If this flag is set, synchronous status latch will be enabled. |
| TPMC150_CONV_ENABLE | If this flag is set, synchronous conversion will be enabled.<br>If the flag is set for channel 1, channel 1 and 2 will be coupled.<br>If the flag is set for channel 3, channel 3 and 4 will be coupled.<br>This flag will be ignored for channel 2 and 4. |

## EXAMPLE

```
#include “tpmc150api.h”

TPMC150_HANDLE     hdl;
TPMC150_STATUS     result;
/*
** configure converter channel 2
**   - resolution: 16bit
**   - no synchronization
*/
result = tpmc150ConfigureConverter(hdl,
                                   2,
                                   16,
                                   0);
if (result != TPMC150_OK)
{
     /* handle error */
}
else
{
     /* configuration succeeded */
}
```

## RETURN VALUE

On success, TPMC150_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

| Error Code | Description |
|---|---|
| TPMC150_ERR_INVALID_HANDLE | The specified device handle is invalid. |
| TPMC150_ERR_INVAL | Invalid input argument specified or NULL pointer referenced. |

## 2.2.3 tpmc150ConfigMultiConvRead

### NAME

tpmc150ConfigMultiConvRead – configure (synchronous) multiple converter read

### SYNOPSIS

```
TPMC150_STATUS tpmc150ConfigMultiConvRead
(
    TPMC150_HANDLE      hdl,
    unsigned int        channels,
    unsigned int        flags
)
```

### DESCRIPTION

This function configures the multiple converter read. The converter channels configured for multiple read will be synchronized. The first read to a multiple read enabled converter channel latches the value/state of the other enabled converter channels.

### PARAMETERS

*hdl*

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

*channels*

This argument specifies the converter channels connected to the multichannel read mode. This value receives a bit-field of the desired channels. Bit 0 corresponds to channel 1, bit 1 corresponds to channel 2 and so on. Using the macro TPMC150_CHAN_ENABLE(*chanNo*) returns the corresponding bit to enable channel *chanNo*. Possible values for *chanNo* are 1 up to the number of available converter channels on the specified device.

*flags*

This argument specifies a set of bit flags that controls specifies configuration:

| Value | Description |
|---|---|
| TPMC150_MULTIREAD_ENABLE | If this flag is set, multi read will be enabled. If this flag is not set, multi read is disabled. |

## EXAMPLE

```
#include “tpmc150api.h”

TPMC150_HANDLE     hdl;
TPMC150_STATUS     result;

/*
** couple converter channel 1, 2 and 4 for synchronous read
*/
result = tpmc150ConfigMultiConvRead(hdl,
                                    TPMC150_CHAN_ENABLE(1) |
                                    TPMC150_CHAN_ENABLE(2) |
                                    TPMC150_CHAN_ENABLE(4),
                                    TPMC150_MULTIREAD_ENABLE);
if (result != TPMC150_OK)
{
     /* handle error */
}
else
{
     /* multiple read configured */
}
```

## RETURN VALUE

On success, TPMC150_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

| Error Code | Description |
|---|---|
| TPMC150_ERR_INVALID_HANDLE | The specified device handle is invalid. |

## 2.2.4 tpmc150ReadEncoder

### NAME

tpmc150ReadEncoder – read value from an encoder channel

### SYNOPSIS

```
TPMC150_STATUS tpmc150ReadEncoder
(
    TPMC150_HANDLE      hdl,
    int                 chanNo,
    unsigned int        *pValue
)
```

### DESCRIPTION

This function reads the value from the specified encoder channel. Depending on the configuration the function will return the current value of the encoder or a value latched by a read of another encoder channel.

### PARAMETERS

*hdl*

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

*chanNo*

This parameter specifies the encoder channel. Allowed values are 1 up to the number of available encoder channels on the specified device.

*pValue*

This parameter points to a buffer, where the function will store the read encoder value.

## EXAMPLE

```
#include “tpmc150api.h”

TPMC150_HANDLE     hdl;
TPMC150_STATUS     result;
unsigned int       encValue;

/*
** read current value of encoder channel 1
*/
result = tpmc150ReadEncoder(hdl,
                            1,
                            &encValue);
if (result != TPMC150_OK)
{
     /* handle error */
}
else
{
     printf(“value: 0x%08X\n”, encValue);
}
```

## RETURN VALUE

On success, TPMC150_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

| Error Code | Description |
|---|---|
| TPMC150_ERR_INVALID_HANDLE | The specified device handle is invalid. |
| TPMC150_ERR_INVAL | Invalid input argument specified or NULL pointer referenced. |

## 2.2.5 tpmc150WriteEncoderPreload

### NAME

tpmc150WriteEncoderPreload – set the preload value of an encoder channel

### SYNOPSIS

```
TPMC150_STATUS tpmc150WriteEncoderPreload
(
    TPMC150_HANDLE      hdl,
    int                 chanNo,
    unsigned int        value,
    unsigned int        flags
)
```

### DESCRIPTION

This function sets the preload value for the specified encoder channel.

### PARAMETERS

*hdl*

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

*chanNo*

This parameter specifies the encoder channel. Allowed values are 1 up to the number of available encoder channels on the specified device.

*value*

This argument specifies the new preload value.

*flags*

This argument specifies a set of bit flags for configuration:

| Value | Description |
|---|---|
| TPMC150_IMMEDIATE_LOAD | If this flag is set, an immediate load will be executed and the new value will be loaded. |

## EXAMPLE

```
#include “tpmc150api.h”

TPMC150_HANDLE     hdl;
TPMC150_STATUS     result;

/*
** set preload value of channel 4 to 0 and immediately load the value
*/
result = tpmc150WriteEncoderPreload(hdl,
                                    4,
                                    0,
                                    TPMC150_IMMEDIATE_LOAD);
if (result != TPMC150_OK)
{
     /* handle error */
}
else
{
     /* preload values set*/;
}
```

## RETURN VALUE

On success, TPMC150_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

| Error Code | Description |
|---|---|
| TPMC150_ERR_INVALID_HANDLE | The specified device handle is invalid. |
| TPMC150_ERR_INVAL | Invalid input argument specified. |

## 2.2.6 tpmc150ConfigureEncoder

### NAME

tpmc150ConfigureEncoder – configure encoder channel

### Synopsis

### SYNOPSIS

```
TPMC150_STATUS tpmc150ConfigureEncoder
(
        TPMC150_HANDLE          hdl,
        int                     chanNo,
        unsigned int            signalMode,
        unsigned int            referenceMode,
        unsigned int            flags
)
```

### DESCRIPTION

This function configures the specified converter channel.

### PARAMETERS

*hdl*

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

*chanNo*

This parameter specifies the encoder channel. Allowed values are 1 up to the number of encoder channels on the specified device.

*signalMode*

The argument specifies the signal mode. Valid signal modes are:

| Signal Mode | Description |
|---|---|
| TPMC150_IO_M_CONESIGANA_OFF | disable counter |
| TPMC150_IO_M_CONESIGANA_1X | 1x - single |
| TPMC150_IO_M_CONESIGANA_2X | 2x - double |
| TPMC150_IO_M_CONESIGANA_4X | 4x - quad |

*referenceMode*

The argument specifies the reference mode. Valid reference modes are:

| Reference Mode | Description |
|---|---|
| TPMC150_IO_M_CONEREF_NONE | None reference mode |
| TPMC150_IO_M_CONEREF_REF | Reference mode |
| TPMC150_IO_M_CONEREF_AUTOREF | Auto reference mode |
| TPMC150_IO_M_CONEREF_INDEX | Index mode |

*flags*

This argument specifies a set of bit flags for configuration:

| Value | Description |
|---|---|
| TPMC150_ENCEMU_ENABLE | If this flag is set, the incremental encoder emulation will be enabled. |
| TPMC150_ENCEMUOUT_ENABLE | If this flag is set, the incremental encoder emulation output signal will be enabled. |

## EXAMPLE

```
#include “tpmc150api.h”

TPMC150_HANDLE     hdl;
TPMC150_STATUS     result;

/*
** configure encoder channel 2
**    4x – quad mode
**    none reference mode
**    encoder emulation and output enabled
*/
result = tpmc150ConfigureEncoder(hdl,
                                 2,
                                 TPMC150_IO_M_CONESIGANA_4X,
                                 TPMC150_IO_M_CONEREF_NONE,
                                 (TPMC150_ENCEMU_ENABLE |
                                  TPMC150_ENCEMUOUT_ENABLE)
                                 );
if (result != TPMC150_OK)
{
     /* handle error */
}
else
{
     /* configuration succeeded */
}
```

## RETURN VALUE

On success, TPMC150_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

| Error Code | Description |
|---|---|
| TPMC150_ERR_INVALID_HANDLE | The specified device handle is invalid. |
| TPMC150_ERR_INVAL | Invalid input argument specified. |

## 2.2.7 tpmc150ConfigMultiEncRead

### NAME

tpmc150ConfigMultiEncRead – configure (synchronous) multiple encoder read

### SYNOPSIS

```
TPMC150_STATUS tpmc150ConfigMultiEncRead
(
    TPMC150_HANDLE      hdl,
    unsigned int        channels,
    unsigned int        flags
)
```

### DESCRIPTION

This function configures the multiple encoder read. The encoder channels configured for multiple read will be synchronized. The first read to a multiple read enabled encoder channel latches the value/state of the other enabled encoder channels.

### PARAMETERS

*hdl*

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

*channels*

This argument specifies the encoder channels connected to the multichannel read mode. This value receives a bit-field of the desired channels. Bit 0 corresponds to channel 1, bit 1 corresponds to channel 2 and so on. Using the macro TPMC150_CHAN_ENABLE(*chanNo*) returns the corresponding bit to enable channel *chanNo*. Possible values for *chanNo* are 1 up to the number of available encoder channels on the specified device.

*flags*

This argument specifies a set of bit flags for configuration:

| Value | Description |
|---|---|
| TPMC150_MULTIREAD_ENABLE | If this flag is set, multi read will be enabled. If this flag is not set, multi read is disabled. |

## EXAMPLE

```
#include “tpmc150api.h”

TPMC150_HANDLE     hdl;
TPMC150_STATUS     result;
unsigned int       in_value;

/*
** couple encoder channel 1, 2 and 4 for synchronous read
*/
result = tpmc150ConfigMultiEncRead(hdl,
                                   TPMC150_CHAN_ENABLE(1) |
                                   TPMC150_CHAN_ENABLE(2) |
                                   TPMC150_CHAN_ENABLE(4),
                                   TPMC150_MULTIREAD_ENABLE);
if (result != TPMC150_OK)
{
     /* handle error */
}
```

## RETURN VALUE

On success, TPMC150_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

| Error Code | Description |
|---|---|
| TPMC150_ERR_INVALID_HANDLE | The specified device handle is invalid. |

## 2.2.8 tpmc150GetDigitialInput

### NAME

tpmc150GetDigitialInput – read state of the digital input lines

### SYNOPSIS

```
TPMC150_STATUS tpmc150GetDigitialInput
(
    TPMC150_HANDLE      hdl,
    unsigned int        *pData
)
```

### DESCRIPTION

This function reads the current state of the digital input lines.

### PARAMETERS

*hdl*

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

*pData*

This parameter points to a buffer, where the function will store the digital input states. A set bit specifies an active and a reset bit a passive input state.
Bit 0 specifies the state on digital input 1, bit 1 the state of digital input 2, bit 2 the state of digital input 3, and bit 3 the state of digital input 4.

## EXAMPLE

```
#include “tpmc150api.h”

TPMC150_HANDLE     hdl;
TPMC150_STATUS     result;
unsigned int       diginVal;

/*
** read digital input state
*/
result = tpmc150GetDigitialInput(hdl, &diginVal);
if (result != TPMC150_OK)
{
     /* handle error */
}
else
{
     printf(“value: 0x%02X\n”, diginVal);
}
```

## RETURN VALUE

On success, TPMC150_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

| Error Code | Description |
|---|---|
| TPMC150_ERR_INVALID_HANDLE | The specified device handle is invalid. |
| TPMC150_ERR_INVAL | Invalid input argument specified or NULL pointer referenced. |

## 2.2.9 tpmc150WaitHighTrans

### NAME

tpmc150WaitHighTrans – wait for a high transition event on digital input line

### SYNOPSIS

```
TPMC150_STATUS tpmc150WaitHighTrans
(
    TPMC150_HANDLE      hdl,
    int                 chanNo,
    int                 msTimeout
)
```

### DESCRIPTION

This function waits until a low-to-high transition on a specified digital input channel occurs.

### PARAMETERS

*hdl*

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

*chanNo*

This parameter specifies the digital input channel where the event shall occur. Allowed values are 1 up to 4.

*msTimeout*

This argument specifies the time (in ms) the function is willing to wait for the event. If the specified time has passed and the event has not occurred, the function will return with an appropriate error code. The function will never timeout if a value of -1 is specified.
The timeout time has resolution of the systems clock. The function will timeout after the specified time has passed and the next clock event occurs.

## EXAMPLE

```
#include “tpmc150api.h”

TPMC150_HANDLE     hdl;
TPMC150_STATUS     result;

/*
** wait for high transition on channel 1, timeout after 10 seconds
*/
result = tpmc150WaitHighTrans(hdl, 1, 10000);
if (result != TPMC150_OK)
{
     /* handle error */
}
else
{
     /* event has occurred */
}
```

## RETURN VALUE

On success, TPMC150_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

| Error Code | Description |
|---|---|
| TPMC150_ERR_INVALID_HANDLE | The specified device handle is invalid. |
| TPMC150_ERR_INVAL | Invalid input argument specified. |
| TPMC150_ERR_TIMEDOUT | Waiting for event has exceeded specified time. |
| TPMC150_ERR_BUSY | Another process is already waiting for an event of this Input line. |

## 2.2.10 tpmc150WaitLowTrans

### NAME

tpmc150WaitLowTrans – wait for a low transition event on digital input line

### SYNOPSIS

```
TPMC150_STATUS tpmc150WaitLowTrans
(
    TPMC150_HANDLE      hdl,
    int                 chanNo,
    int                 msTimeout
)
```

### DESCRIPTION

This function waits until a high-to-low transition on a specified digital input channel occurs.

### PARAMETERS

*hdl*

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

*chanNo*

This parameter specifies the digital input channel where the event shall occur. Allowed values are 1 up to 4.

*msTimeout*

This argument specifies the time (in ms) the function is willing to wait for the event. If the specified time has passed and the event has not occurred, the function will return with an appropriate error code.
The timeout time has resolution of the systems clock. The function will timeout after the specified time has passed and the next clock event occurs.

## EXAMPLE

```
#include “tpmc150api.h”

TPMC150_HANDLE     hdl;
TPMC150_STATUS     result;

/*
** wait for low transition on channel 1, timeout after 10 seconds
*/
result = tpmc150WaitLowTrans(hdl, 1, 10000);
if (result != TPMC150_OK)
{
     /* handle error */
}
else
{
     /* event has occurred */
}
```

## RETURN VALUE

On success, TPMC150_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

| Error Code | Description |
|---|---|
| TPMC150_ERR_INVALID_HANDLE | The specified device handle is invalid. |
| TPMC150_ERR_INVAL | Invalid input argument specified. |
| TPMC150_ERR_TIMEDOUT | Waiting for event has exceeded specified time. |
| TPMC150_ERR_BUSY | Another process is already waiting for an event of this Input line. |

# 3 Legacy I/O System Functions

This chapter describes the legacy 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.

**The legacy I/O system functions are only relevant for the legacy TPMC150 driver. For the VxBus-enabled TPMC150 driver, the driver will be installed automatically in the I/O system and devices will be created as needed for detected modules.**

## 3.1 tpmc150PciInit

### NAME

tpmc150PciInit() – Generic PCI device initialization

### SYNOPSIS

void tpmc150PciInit()

### DESCRIPTION

This function is required only for Intel x86 VxWorks platforms. The purpose is to setup the MMU mapping for all required TPMC150 PCI spaces (base address register) and to enable the TPMC150 device for access.

The global variable *tpmc150Status* 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 tpmc150Status is equal to the number of mapped PCI spaces |
| 0 | No TPMC150 device found |
| < 0 | Initialization failed. The value of (tpmc150Status & 0xFF) is equal to the number of mapped spaces until the error occurs.<br>Possible cause: Too few entries for dynamic mappings in sysPhysMemDesc[].<br>Remedy: Add dummy entries as necessary (syslib.c). |

### EXAMPLE

```
extern void tpmc150PciInit();

tpmc150PciInit();
```

# 4 Appendix

## 4.1 Enable RTP-Support

Using TPMC150 devices tunneled from Real Time Processes (RTPs) is implemented. For this the “TEWS TPMC150 IOCTL command validation” must be enabled in system configuration.

The API source file “tpmc150api.c” must be added to the RTP-Project directory and built together with the RTP-application.

The definition of TVXB_RTP_CONTEXT must be added to the project, which is used to eliminate kernel headers, values and functions from the used driver files.

Find more detailed information in “TEWS TECHNOLOGIES VxWorks Device Drivers - Installation Guide”.

**All legacy functions, functions for version compatibility and debugging functions are not usable from RTPs.**

## 4.2 Debugging and Diagnostic

The TPMC150 device driver provides a function and debug statements to display versatile information of the driver installation and status on the debugging console.

If the VxBus driver is used, the TPMC150 show routine is included in the driver by default and can be called from the VxWorks shell. If this function is not needed or program space is rare the function can be removed from the code by un-defining the macro INCLUDE_TPMC150_SHOW in tpmc150drv.c

The tpmc150Show function (only if VxBus is used) displays detailed information about probed modules, assignment of devices respective device names to probed TPMC150 modules and device statistics.

If TPMC150 modules were probed but no devices were created it may be helpful to enable debugging code inside the driver code by defining the macro TPMC150_DEBUG in tpmc150drv.c.

**In contrast to VxBus TPMC150 devices, legacy TPMC150 devices must be created "manually". This will be done with the first call to the tpmc150Open API function.**

```
-> tpmc150Show
Probed Modules:
   [0] TPMC150-10: Bus=4, Dev=2, DevId=0x0096, VenId=0x1498, Init=OK, vxDev=0xffff800000004f20

Associated Devices:
   [0] TPMC150-10: /tpmc150/0

Device Statistics:
   /tpmc150/0:
        open count = 0
        interrupt count = 0
        interrupt count = 0
          event on digIn[1] = 0
        interrupt count = 0
          event on digIn[2] = 0
        interrupt count = 0
          event on digIn[3] = 0
        interrupt count = 0
          event on digIn[4] = 0
value = 32 = 0x20 = ' '
```