The Embedded I/O Company

TEWS TECHNOLOGIES

TPMC680-SW-82

Linux Device Driver

64 Digital Input/Output

Version 2.0.x

User Manual

Issue 2.0.1

November 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

TPMC680-SW-82

Linux Device Driver

64 Digital Input/Output

Supported Modules:
TPMC680-10

Issue	Description	Date
1.0	First Issue	April 9, 2003
1.1.0	Kernel 2.6.x Revision	April 8, 2005
1.1.1	Description of installation revised	January 29, 2006
1.1.2	File list modified, new address TEWS LLC, general revision	November 9, 2006
1.1.3	address TEWS LLC removed	April 1, 2010
2.0.0	New chapter for API functions, chapter Device Input/Output functions removed	July 25, 2011
2.0.1	GPL-File (COPYING.txt) added	Novemver 15, 2021

Table of Contents

1 Introduction

The TPMC680-SW-82 Linux device driver allows the operation of a TPMC680 digital I/O PMC on Linux operating systems.

Supported features:

- read digital input value (8 bit / 64 bit ports)
- write digital output value(8 bit / 64 bit ports)
- receive and transmit parallel data (16 bit / 32 bit handshake ports)
- configure port size, direction and handshake mode
- wait for a specified input event (8 bit / 64 bit ports)

The TPMC680-SW-82 device driver supports the modules listed below:

TPMC680-10	8 x 8 Bit Digital Inputs/Outputs (5V TTL)	(PMC)

To get more information about the features and usage of TPMC680 devices it is recommended to read the manuals listed below.

TPMC680 User Manual
TPMC680 Engineering Manual

2 Installation

The directory TPMC680-SW-82 on the distribution media contains the following files:

TPMC680-SW-82-2.0.1.pdf	This manual in PDF format
TPMC680-SW-82-SRC.tar.gz	GZIP compressed archive with driver source code
Release.txt	Release information
ChangeLog.txt	Release history

The GZIP compressed archive TPMC680-SW-82-SRC.tar.gz contains the following files and directories:

Directory path 'tpmc680':

tpmc680.c	Driver source code
tpmc680def.h	Driver private include file
tpmc680.h	Driver public include file for application program
Makefile	Device driver make file
makenode	Script to create device nodes on the file system
COPYING	Copy of the GNU Public License (GPL)
api/tpmc680api.c	API source file
api/tpmc680api.h	API include file
include/tpxxxhwdep.c	Low level hardware access functions source file
include/tpxxxhwdep.h	Access functions header file
include/tpmodule.c	Driver independent library
include/tpmodule.h	Driver independent library header file
example/tpmc680exa.c	Example application
example/Makefile	Example application make file

In order to perform an installation, extract all files of the archive TPMC680-SW-82-SRC.tar.gz to the desired target directory. The command 'tar -xzvf TPMC680-SW-82-SRC.tar.gz' will extract the files into the local directory.

- Login as *root* and change to the target directory
- Copy tpmc680.h to */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 tpmc680drv

- 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 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 TPMC680 module found. The first TPMC680 module can be accessed with device node /dev/tpmc680_0, the second with device node /dev/tpmc680_1 and so on.

The assignment of device nodes to physical TPMC680 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 tpmc680drv

If your kernel has enabled devfs or sysfs (udev), all /dev/tpmc680_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 "*tpmc680drv: 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 dynamic device file system installed. The TPMC680 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 tpmc680def.h, change the following symbol to appropriate value and enter `make install` to create a new driver.

TPMC680_MAJOR	Valid numbers are in range between 0 and 255. A value of 0 means dynamic number allocation.

Example:

```
#define TPMC680_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 API Documentation

3.1 General Functions

3.1.1 tpmc680Open

NAME

tpmc680Open – Opens a Device

SYNOPSIS

```
TPMC680_HANDLE tpmc680Open
(
     char *DeviceName
);
```

DESCRIPTION

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

PARAMETERS

DeviceName

This parameter points to a null-terminated string that specifies the name of the device.

EXAMPLE

```
#include “tpmc680api.h”

TPMC680_HANDLE hdl;

/*
** open file descriptor to device
*/
hdl = tpmc680Open( “/dev/tpmc680_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.

3.1.2 tpmc680Close

NAME

tpmc680Close – Closes a Device

SYNOPSIS

```
TPMC680_STATUS tpmc680Close
(
     TPMC680_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 “tpmc680api.h”

TPMC680_HANDLE hdl;
TPMC680_STATUS result;

/*
** close file descriptor to device
*/
result = tpmc680Close ( hdl );
if (result != TPMC680_OK)
{
     /* handle close error */
}
```

RETURNS

On success TPMC680_OK, or an appropriate error code.

ERROR CODES

TPMC680_ERR_INVALID_HANDLE	The specified TPMC680_HANDLE is invalid.

Other returned error codes are system error conditions.

3.2 Device Access Functions

3.2.1 tpmc680SetPortMode

NAME

tpmc680SetPortMode – Configure port

SYNOPSIS

```
TPMC680_STATUS tpmc680SetPortMode
(
    TPMC680_HANDLE          hdl,
    unsigned int            portNo,
    unsigned int            portSize,
    unsigned int            portDirection,
    unsigned int            handshakeMode,
    unsigned int            handshakeFifoLevelMode
);
```

DESCRIPTION

This function configures the specified port of the TPMC680. The function sets size, direction and handshake modes. If port sizes greater 8 bit are used some (hardware) ports will be concatenated to a (software) port which is responsible to control the I/O function. Mainly responsible for port concatenations are port 0 and 2. Port 0 can be used for 16 and 32 bit handshake and 64 bit synchronous I/O. Port 2 can be used for 16 bit handshake I/O.

The table below shows to which port number the (hardware) ports will be assigned at the possible configurations of ports 0 and 2.

(Hardware) Port	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
(Software) Port number	7 (8 bit)	6 (8 bit)	5 (8 bit)	4 (8 bit)	3 (8 bit)	2 (8 bit)	1 (8 bit)	0 (8 bit)
	7 (8 bit)	6 (8 bit)	5 (8 bit)	4 (8 bit)	3 (8 bit)	2 (8 bit)	0 (16 bit / HS)	
	7 (8 bit)	6 (8 bit)	5 (8 bit)	4 (8 bit)	2 (16 bit / HS)		1 (8 bit)	0 (8 bit)
	7 (8 bit)	6 (8 bit)	5 (8 bit)	4 (8 bit)	2 (16 bit / HS)		0 (16 bit / HS)	
	7 (8 bit)	6 (8 bit)	5 (8 bit)	4 (8 bit)	0 (32 bit / HS)			
	0 (64 bit / synchronous)							

Additionally to the port concatenations the direction of port 4 and port 5 may be changed if port 0 or port 2 is used in handshake mode. Port 4 will be configured as input port and port 5 may be configured for output. Bit 0 and 1 will be reserved for the handshake signals and are not anymore controlled by the ports.

Please also refer to the TPMC680 User Manual to get more information about the port configuration and use signals.

Changing a port size from a bigger to a smaller size will also change the mode of the connected ports. The ports will be set to 8 bit mode and they will keep the configured direction.

PARAMETERS

hdl

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

portNo

This argument specifies the port that shall be configured. Valid values are between 0 and 7.

portSize

This argument specifies the size of the port. The following table describes the allowed port sizes and for which ports they are allowed.

Value	Ports	Description
TPMC680_MODE_SIZE_8BIT	0, 1, 2, 3, 4, 5, 6, 7	The port has a width of 8 bit. Each port can be accessed separately.
TPMC680_MODE_SIZE_16BIT	0,2	The port has a width of 16 bit and the output is controlled by the handshake signals. Two ports are used together. If port 0 is selected port 1 is used also. If port 2 is selected also port 3 will be used. The configuration of the connected ports is always adapted. **If this mode is selected for any port the handshake port 4 will be configured as an 8-bit input port.**
TPMC680_MODE_SIZE_32BIT	0	The port has a width of 32 bit and the output is controlled by the handshake signals. The ports 0, 1, 2 and 3 will be used together. The configuration of the connected ports is always set together. **If this mode is selected the handshake port 4 will be configured as an 8-bit input port.**
TPMC680_MODE_SIZE_64BIT	0	All ports are connected and can be used as simple 64 bit input or output port. All ports get the same configuration.

portDirection

This argument specifies the direction of the port. All connected ports will be set to the same direction. Allowed values are:

Value	Description
TPMC680_MODE_DIR_INPUT	The port will be used as an input port.
TPMC680_MODE_DIR_OUTPUT	The port will be used as an output port.

handshakeMode

This argument specifies the handshake mode and is only valid if the port is configured for 16 or 32 bit mode (*TPMC680_MODE_SIZE_16BIT, TPMC680_MODE_SIZE_32BIT*). Using an output handshake, will change the direction of port 5 to output. The allowed values are:

Value	Description
TPMC680_MODE_HSFLAG_NO	No output handshake will be used.
TPMC680_MODE_HSFLAG_INTERLOCKED	The interlocked output handshake mode will be used.
TPMC680_MODE_HSFLAG_PULSED	The pulsed output handshake mode will be used.

handshakeFifoLevelMode

This argument specifies the handshake event depending on the handshake FIFO fill level. This value is only used if an output handshake is configured. Allowed values are:

Value	Description
TPMC680_MODE_HSFIFOEV_NOTFULL	The event announces FIFO is not full.
TPMC680_MODE_HSFIFOEV_EMPTY	The event announces FIFO is empty.

EXAMPLE

```
#include “tpmc680api.h”

TPMC680_HANDLE     hdl;
TPMC680_STATUS     result;

/*
** Configure port (2)
**        Size: 16-bit
**        Direction: output
**        handshake: interlocked / output event on empty FIFO
*/
result = tpmc680SetPortMode (     hdl,
                                  2,
                                  TPMC680_MODE_SIZE_16BIT,
                                  TPMC680_MODE_DIR_OUTPUT,
                                  TPMC680_MODE_HSFLAG_INTERLOCKED,
                                  TPMC680_MODE_HSFIFOEV_EMPTY);
if (result != TPMC680_OK)
{
     /* handle error */
}
```

RETURNS

On success, TPMC680_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

TPMC680_ERR_INVALID_HANDLE	The specified TPMC680_HANDLE is invalid.
TPMC680_ERR_INVAL	An argument contains an invalid value.
TPMC680_ERR_CHRNG	An invalid port number has been specified.
TPMC680_ERR_ACCESS	The specified port configuration is not allowed.

Other returned error codes are system error conditions.

3.2.2 tpmc680ReadPort

NAME

tpmc680ReadPort – Read state of 8-bit port

SYNOPSIS

```
TPMC680_STATUS tpmc680ReadPort
(
    TPMC680_HANDLE          hdl,
    unsigned int            portNo,
    unsigned char           *pPortVal
);
```

DESCRIPTION

This function reads the current state of the input lines of an 8 bit port on the TPMC680.

The port must be configured in 8 bit mode, otherwise the function will fail.

PARAMETERS

hdl

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

portNo

This argument specifies the port that shall be read. Valid values are between 0 and 7.

pPortVal

This pointer points to an unsigned char where the current state of the port will be stored to.

EXAMPLE

```
#include “tpmc680api.h”

TPMC680_HANDLE     hdl;
TPMC680_STATUS     result;
unsigned char      portState;

/*
** Read from 8-bit port (2)
*/
result = tpmc680ReadPort (   hdl,
                             2,
                             &portState);
if (result == TPMC680_OK)
{
     printf(“Port2: 0x%02X\n”, portState);
}
else
{
     /* handle error */
}
```

RETURNS

On success, TPMC680_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

TPMC680_ERR_INVALID_HANDLE	The specified TPMC680_HANDLE is invalid.
TPMC680_ERR_INVAL	A specified pointer is NULL.
TPMC680_ERR_CHRNG	An invalid port number has been specified.
TPMC680_ERR_ACCESS	Access not allowed with current port configuration.

Other returned error codes are system error conditions.

3.2.3 tpmc680WritePort

NAME

tpmc680WritePort – Write new output value to 8-bit port

SYNOPSIS

```
TPMC680_STATUS tpmc680WritePort
(
    TPMC680_HANDLE          hdl,
    unsigned int            portNo,
    unsigned char           portVal
);
```

DESCRIPTION

This function writes a new output value to an 8 bit port of the TPMC680.

The port must be configured in 8 bit output mode, otherwise the function will fail.

PARAMETERS

hdl

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

portNo

This argument specifies the port that shall be read. Valid values are between 0 and 7.

portVal

This argument specifies the new output value.

EXAMPLE

```
#include “tpmc680api.h”

TPMC680_HANDLE     hdl;
TPMC680_STATUS     result;

/*
** Set 8-bit port (2) (new value 12(hex))
*/
result = tpmc680WritePort (  hdl,
                             2,
                             0x12);
if (result != TPMC680_OK)
{
     /* handle error */
}
```

RETURNS

On success, TPMC680_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

TPMC680_ERR_INVALID_HANDLE	The specified TPMC680_HANDLE is invalid.
TPMC680_ERR_CHRNG	An invalid port number has been specified.
TPMC680_ERR_ACCESS	Access not allowed with current port configuration.

Other returned error codes are system error conditions.

3.2.4 tpmc680ReadPort64

NAME

tpmc680ReadPort64 – Read state of 64-bit port

SYNOPSIS

```
TPMC680_STATUS tpmc680ReadPort64
(
    TPMC680_HANDLE          hdl,
    unsigned int            *pPortVal0_31,
    unsigned int            *pPortVal32_63
);
```

DESCRIPTION

This function reads the current state of the input lines of the 64 bit port on the TPMC680.

The port must be configured in 64 bit mode, otherwise the function will fail.

PARAMETERS

hdl

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

pPortVal0_31

This pointer points to an unsigned int (32-bit) where the current state of the ports 0...3 will be stored to. Port 0 will be stored to bits 0...7, Port 1 to bits 8…15, and so on.

pPortVal32_63

This pointer points to an unsigned int (32-bit) where the current state of the ports 4...7 will be stored to. Port 4 will be stored to bits 0...7, Port 5 to bits 8…15, and so on.

EXAMPLE

```
#include “tpmc680api.h”

TPMC680_HANDLE     hdl;
TPMC680_STATUS     result;
unsigned int       portStateLow;
unsigned int       portStateHigh;

/*
** Read from 64-bit port
*/
result = tpmc680ReadPort64(  hdl,
                             &portStateLow,
                             &portStateHigh);
if (result == TPMC680_OK)
{
     printf(“Port7..0: 0x%08X%08X\n”, portStateHigh, portStateLow);
}
else
{
     /* handle error */
}
```

RETURNS

On success, TPMC680_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

TPMC680_ERR_INVALID_HANDLE	The specified TPMC680_HANDLE is invalid.
TPMC680_ERR_INVAL	A specified pointer is NULL.
TPMC680_ERR_ACCESS	Access not allowed with current port configuration.

Other returned error codes are system error conditions.

3.2.5 tpmc680WritePort64

NAME

tpmc680WritePort64 – Write new output value to 64-bit port

SYNOPSIS

```
TPMC680_STATUS tpmc680WritePort64
(
    TPMC680_HANDLE          hdl,
    unsigned int            portVal0_31,
    unsigned int            portVal32_63,
);
```

DESCRIPTION

This function writes a new output value to the 64 bit port of the TPMC680.

The port must be configured in 64 bit output mode, otherwise the function will fail.

PARAMETERS

hdl

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

portVal0_31

This argument specifies the new output value of the ports 0...3 will be stored to. Port 0 will be stored to bits 0...7, Port 1 to bits 8…15, and so on.

portVal32_63

This argument specifies the new output value of the ports 4...7 will be stored to. Port 4 will be stored to bits 0...7, Port 5 to bits 8…15, and so on.

EXAMPLE

```
#include “tpmc680api.h”

TPMC680_HANDLE     hdl;
TPMC680_STATUS     result;

/*
** Set 64-bit port (new value 7766554433221100(hex))
*/
result = tpmc680WritePort64 (    hdl,
                                 0x33221100,
                                 0x77665544);
if (result != TPMC680_OK)
{
     /* handle error */
}
```

RETURNS

On success, TPMC680_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

TPMC680_ERR_INVALID_HANDLE	The specified TPMC680_HANDLE is invalid.
TPMC680_ERR_ACCESS	Access not allowed with current port configuration.

Other returned error codes are system error conditions.

3.2.6 tpmc680Receive16

NAME

tpmc680Receive16 – Read data received on 16-bit port

SYNOPSIS

```
TPMC680_STATUS tpmc680Receive16
(
    TPMC680_HANDLE      hdl,
    unsigned int        portNo,
    unsigned int        bufSize,
    unsigned short      *pBuf,
    unsigned int        *pValidData
);
```

DESCRIPTION

This function reads data that has been received on a 16 bit input port of the TPMC680.

The port must be configured in 16 bit input mode, otherwise the function will fail.

PARAMETERS

hdl

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

portNo

This argument specifies the port that shall be read. Valid values are 0 and 2.

bufSize

This argument specifies the number data words (16 bit) which can be copied into the input buffer.

pBuf

This pointer points to the input buffer where the received data will be stored to.

pValidData

This pointer points to an unsigned int value where the number of received (valid) data values will be stored to.

EXAMPLE

```
#include “tpmc680api.h”

#define BUFSIZE    5

TPMC680_HANDLE     hdl;
TPMC680_STATUS     result;
unsigned short     inBuf[BUFSIZE];
unsigned int       numData;

/*
** Read received data from 16-bit port (2)
*/
result = tpmc680Receive16 (  hdl,
                             2,
                             BUFSIZE,
                             inBuf,
                             &numData);
if (result == TPMC680_OK)
{
     for (i = 0; i < numData; i++)
          printf(“[%d] 0x%04X\n”, i, inBuf[i]);
}
else
{
     /* handle error */
}
```

RETURNS

On success, TPMC680_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

TPMC680_ERR_INVALID_HANDLE	The specified TPMC680_HANDLE is invalid.
TPMC680_ERR_INVAL	A specified pointer is NULL.
TPMC680_ERR_CHRNG	An invalid port number has been specified.
TPMC680_ERR_ACCESS	Access not allowed with current port configuration.

Other returned error codes are system error conditions.

3.2.7 tpmc680Send16

NAME

tpmc680Send16 – Send data on 16-bit port

SYNOPSIS

```
TPMC680_STATUS tpmc680Send16
(
    TPMC680_HANDLE          hdl,
    unsigned int            portNo,
    unsigned int            bufSize,
    unsigned short          *pBuf,
    unsigned int            *pSentData
);
```

DESCRIPTION

This function sends data on a 16 bit port of the TPMC680. The function places the data into a FIFO and starts transmission. It will not wait until data is send.

The port must be configured in 16 bit output mode, otherwise the function will fail.

PARAMETERS

hdl

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

portNo

This argument specifies the port that shall be used. Valid values are 0 and 2.

bufSize

This argument specifies the number data words (16 bit) in the output buffer.

pBuf

This pointer points to the output buffer containing the data ready to send.

pSentData

This pointer points to an unsigned int value where the number of successfully sent data values will be stored to.

EXAMPLE

```
#include “tpmc680api.h”

#define BUFSIZE    5

TPMC680_HANDLE     hdl;
TPMC680_STATUS     result;
unsigned short     outBuf[BUFSIZE] = {0x1111,0x2222,0x3333,0x4444,0x5555};
unsigned int       numData;

/*
** Read received data from 16-bit port (2)
*/
result = tpmc680Send16 (     hdl,
                             2,
                             BUFSIZE,
                             outBuf,
                             &numData);
if (result != TPMC680_OK)
{
     /* handle error */
}
```

RETURNS

On success, TPMC680_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

TPMC680_ERR_INVALID_HANDLE	The specified TPMC680_HANDLE is invalid.
TPMC680_ERR_INVAL	A specified pointer is NULL.
TPMC680_ERR_CHRNG	An invalid port number has been specified.
TPMC680_ERR_ACCESS	Access not allowed with current port configuration.

Other returned error codes are system error conditions.

3.2.8 tpmc680Receive32

NAME

tpmc680Receive32 – Read data received on 32-bit port

SYNOPSIS

```
TPMC680_STATUS tpmc680Receive32
(
    TPMC680_HANDLE      hdl,
    unsigned int        bufSize,
    unsigned int        *pBuf,
    unsigned int        *pValidData
);
```

DESCRIPTION

This function reads data that has been received on the 32 bit input port of the TPMC680.

The port must be configured in 32 bit input mode, otherwise the function will fail.

PARAMETERS

hdl

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

bufSize

This argument specifies the number data words (32 bit) which can be copied into the input buffer.

pBuf

This pointer points to the input buffer where the received data will be stored to.

pValidData

This pointer points to an unsigned int value where the number of received (valid) data values will be stored to.

EXAMPLE

```
#include “tpmc680api.h”

#define BUFSIZE    5

TPMC680_HANDLE     hdl;
TPMC680_STATUS     result;
unsigned int       inBuf[BUFSIZE];
unsigned int       numData;

/*
** Read received data from 32-bit port (2)
*/
result = tpmc680Receive32 (  hdl,
                             BUFSIZE,
                             inBuf,
                             &numData);
if (result == TPMC680_OK)
{
     for (i = 0; i < numData; i++)
          printf(“[%d] 0x%08X\n”, i, inBuf[i]);
}
else
{
     /* handle error */
}
```

RETURNS

On success, TPMC680_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

TPMC680_ERR_INVALID_HANDLE	The specified TPMC680_HANDLE is invalid.
TPMC680_ERR_INVAL	A specified pointer is NULL.
TPMC680_ERR_ACCESS	Access not allowed with current port configuration.

Other returned error codes are system error conditions.

3.2.9 tpmc680Send32

NAME

tpmc680Send32 – Send data on 32-bit port

SYNOPSIS

```
TPMC680_STATUS tpmc680Send32
(
    TPMC680_HANDLE          hdl,
    unsigned int            portNo,
    unsigned int            bufSize,
    unsigned int            *pBuf,
    unsigned int            *pSentData
);
```

DESCRIPTION

This function sends data on the 32 bit port of the TPMC680. The function places the data into a FIFO and starts transmission. It will not wait until data is send.

The port must be configured in 32 bit output mode, otherwise the function will fail.

PARAMETERS

hdl

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

portNo

This argument specifies the port that shall be used. Valid values are 0 and 2.

bufSize

This argument specifies the number data words (32 bit) in the output buffer.

pBuf

This pointer points to the output buffer containing the data ready to send.

pSentData

This pointer points to an unsigned int value where the number of successfully sent data values will be stored to.

EXAMPLE

```
#include “tpmc680api.h”

#define BUFSIZE    3

TPMC680_HANDLE     hdl;
TPMC680_STATUS     result;
unsigned int       outBuf[BUFSIZE] = {0x11112222,0x33334444,0x55556666};
unsigned int       numData;

/*
** Send data on 32-bit port
*/
result = tpmc680Send32 (     hdl,
                             BUFSIZE,
                             outBuf,
                             &numData);
if (result != TPMC680_OK)
{
     /* handle error */
}
```

RETURNS

On success, TPMC680_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

TPMC680_ERR_INVALID_HANDLE	The specified TPMC680_HANDLE is invalid.
TPMC680_ERR_INVAL	A specified pointer is NULL.
TPMC680_ERR_ACCESS	Access not allowed with current port configuration.

Other returned error codes are system error conditions.

3.2.10 tpmc680WaitForEvent

NAME

tpmc680WaitForEvent – Wait for a specified input event

SYNOPSIS

```
TPMC680_STATUS tpmc680WaitForEvent
(
    TPMC680_HANDLE      hdl,
    unsigned int        portNo,
    unsigned int        lineNo,
    unsigned int        transition,
    unsigned int        timeout
);
```

DESCRIPTION

This function waits for a specified event on a specified input line of the TPMC680.

The port must be configured in 8 bit or 64 bit input mode, otherwise the function will fail.

PARAMETERS

hdl

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

portNo

This argument specifies the port. Valid values are between 0 and 7.

lineNo

This argument specifies the ports line number. Valid values are between 0 and 7.

transition

This argument specifies the transition event to wait for. The following events are supported:

Value	Description
TPMC680_IO_EDGE_HI	The event will occur if the specified input line changes from Low to High.
TPMC680_IO_EDGE_LO	The event will occur if the specified input line changes from High to Low.
TPMC680_IO_EDGE_ANY	The event will occur if the specified input line changes its value.

timeout

This argument specifies the timeout in milliseconds. If the specified event does not occur in the specified time, the function will return with an error code.

EXAMPLE

```
#include “tpmc680api.h”

TPMC680_HANDLE     hdl;
TPMC680_STATUS     result;

/*
** Wait for a high to low transition on line 5 of port 6
**        Timeout after 10000 milliseconds
*/
result = tpmc680WaitForEvent (    hdl,
                                  6,
                                  5,
                                  TPMC680_IO_EDGE_LO,
                                  10000);
if (result != TPMC680_OK)
{
     /* handle error */
}
```

RETURNS

On success, TPMC680_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

TPMC680_ERR_INVALID_HANDLE	The specified TPMC680_HANDLE is invalid.
TPMC680_ERR_INVAL	A specified argument contains an invalid value.
TPMC680_ERR_CHRNG	An invalid port number has been specified.
TPMC680_ERR_ACCESS	Access not allowed with current port configuration.
TPMC680_ERR_BUSY	There is already an active job waiting for an event on the specified input line.
TPMC680_ERR_TIMEOUT	The function timed out

Other returned error codes are system error conditions.

4 Diagnostic

If the TPMC680 does not work properly it is helpful to get some status information from the driver respective kernel.

The Linux */proc* file system provides information about kernel, resources, driver, devices and so on. The following screen dumps displays information of a correct running TPMC680 driver (see also the proc man pages).

```
# lspci -v
  …
04:02.0 Signal processing controller: TEWS Technologies GmbH Device 02a8
        Subsystem: TEWS Technologies GmbH Device 000a
        Flags: medium devsel, IRQ 17
        Memory at feb9f400 (32-bit, non-prefetchable) [size=128]
        I/O ports at e800 [size=128]
        Memory at feb9f000 (32-bit, non-prefetchable) [size=256]
        Kernel driver in use: TEWS TECHNOLOGIES - TPMC680 64 Digital IO -
        Kernel modules: tpmc680drv
  …


# cat /proc/devices
Character devices:
  …
250 tpmc680drv
  …

cat /proc/iomem
00000000-0000ffff : reserved
00010000-0009fbff : System RAM
  …
80000000-ffffffff : PCI Bus 0000:00
  …
  feb00000-febfffff : PCI Bus 0000:04
    feb9f000-feb9f0ff : 0000:04:02.0
      feb9f000-feb9f0ff : TPMC680
  …
```