The Embedded I/O Company

TEWS TECHNOLOGIES

TPMC680-SW-95

QNX6 - Neutrino Device Driver

64 Digital Inputs/Outputs

Version 1.0.0

User Manual

Issue 1.0.0

June 2005

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

TEWS TECHNOLOGIES LLC
1 E. Liberty Street, Sixth Floor
Reno, Nevada 89504 / USA
Phone: +1 (775) 686 6077
Fax: +1 (775) 686 6024
e-mail: usasales@tews.com
www.tews.com

TPMC680-SW-95

64 Digital Inputs/Outputs

QNX6 - Neutrino Device Driver

This document contains information, which is proprietary to TEWS TECHNOLOGIES GmbH. Any reproduction without written permission is forbidden.

TEWS TECHNOLOGIES GmbH has made any effort to ensure that this manual is accurate and complete. However TEWS TECHNOLOGIES GmbH reserves the right to change the product described in this document at any time without notice.

TEWS TECHNOLOGIES GmbH is not liable for any damage arising out of the application or use of the device described herein.

Issue	Description	Date
1.0.0	First Issue	June 1, 2005

Table of Content

1 Introduction

The TPMC680-SW-95 QNX-Neutrino device driver allows the operation of a TPMC680 64 Input/Outputs PMC on QNX-Neutrino operating systems.

The TPMC680 device driver is basically implemented as a user installable Resource Manager. The standard file (I/O) functions (open, close and devctl) provide the basic interface for opening and closing a file descriptor and for performing device I/O and control operations.

The TPMC680 device driver includes the following functions:

- Configuring ports to work as 8-, 16-, 32- and 64 bit ports.
- Configuring port direction
- Setting I/O line output in 8- and 64-bit configuration
- Getting I/O line input in 8- and 64-bit configuration
- Receiving data via 16- or 32-bit handshake ports
- Transmitting data via 16- or 32-bit handshake ports
- Wait for input events on I/O lines of 8- and 64-bit ports

To understand all features of this device driver, it is very important to read TPMC680 User Manual.

2 Installation

The distribution media contains the following files:

TPMC680-SW-95-SRC.tar	tar-archieve containing driver and example files
TPMC680-SW-95.pdf	this manual
Release.txt	Information about the Device Driver Release

Following files are stored in the tar-archive:

/driver/tpmc680.c	Driver source code
/driver/tpmc680.h	Definitions and data structures for driver and application
/driver/tpmc680def.h	Device driver include
/driver/node.c	Queue management source code
/driver/node.h	Queue management definitions
/driver/nto/*	Build path
/example/example.c	Example application
/example/nto/*	Build path

For installation copy the tar-archive into the /usr/src directory and unpack it (e.g. tar –xvf TPMC680-SW-95-SRC.tar). After that the necessary directory structure for the automatic build and the source files are available underneath the new directory called tpmc680.

It is absolutely important to extract the TPMC680 tar archive in the /usr/src directory. Otherwise the automatic build with make will fail.

2.1 Build the device driver

Change to the /usr/src/tpmc680/driver directory

Execute the Makefile:

```
# make install
```

After successful completion the driver binary (tpmc680) will be installed in the /bin directory.

Build the example application

Change to the */usr/src/tpmc680/example* directory

Execute the Makefile:

```
# make install
```

After successful completion the example binary (***tp680exam***) will be installed in the /bin directory.

2.2 Start the driver process

To start the TPMC310 device driver respective you have to enter the process name with optional parameter from the command shell or in the startup script.

```
tpmc680 [-v] &
```

The TPMC680 Resource Manager registers created devices in the Neutrinos pathname space under following names.

```
/dev/tpmc680_0
/dev/tpmc680_1
…
/dev/tpmc680_x
```

This pathname must be used in the application program to open a path to the desired TPMC680 device.

```
fd = open("/dev/tpmc680_0", O_RDWR);
```

For debugging, you can start the TPMC680 Resource Manager with the –v option. Now the Resource Manager will print versatile information about TPMC680 configuration and command execution on the terminal window.

```
tpmc680 –v &
```

2.3 Receive and Transmit FIFO Configuration

The size of receive and transmit FIFO can be configured in *tpmc680def.h*. The values of the following definition can be adapted.

TP680_IOBUFSIZE16

Defines the depth of the FIFOs for port 0 and port 2 used for 16-bit handshake mode. The value specifies the number 16-bit words.

TP680_IOBUFSIZE32

Defines the depth of the FIFO for port 0 used for 32-bit handshake mode. The value specifies the number 32-bit words.

After changing any of the values, the driver has to be rebuilt.

2.4 Application interface configuration

The size of read and write buffers can be configured in *tpmc680.h*. The values of the following definition can be adapted.

TP680_MAX_16BIT_ELEM

Defines the maximum number of 16-bit words, that can be read or written at once.

TP680_MAX_32BIT_ELEM

Defines the maximum number of 32-bit words, that can be read or written at once.

After changing any of the values, the driver and application has to be rebuilt.

3 Device Input/Output functions

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

3.1 open()

NAME

open() - open a file descriptor

SYNOPSIS

#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int open (const char *pathname, int flags)

DESCRIPTION

The *open* function creates and returns a new file descriptor for the TPMC680 named by pathname. The flags argument controls how the file is to be opened. TPMC680 devices must be opened O_RDWR.

EXAMPLE

```
int fd;

fd = open(“/dev/tpmc680_0”, O_RDWR);
```

RETURNS

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

ERRORS

Returns only Neutrino specific error codes, see Neutrino Library Reference.

SEE ALSO

Library Reference - open()

3.2 close()

NAME

close() – close a file descriptor

SYNOPSIS

#include <unistd.h>

int close (int filedes)

DESCRIPTION

The close function closes the file descriptor *filedes*.

EXAMPLE

```
int fd;

...

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

RETURNS

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

ERRORS

Returns only Neutrino specific error code, see Neutrino Library Reference.

SEE ALSO

Library Reference - close()

3.3 devctl()

NAME

devctl() – device control functions

SYNOPSIS

```
#include <sys/types.h>
#include <unistd.h>
#include <devctl.h>

int devctl
(
        int         filedes,
        int         dcmd,
        void        *data_ptr,
        size_t      n_bytes,
        int         *dev_info_ptr
)
```

DESCRIPTION

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

The argument *dcmd* specifies the control code for the operation.

The arguments *data_ptr* and *n_bytes* depends on the command and will be described for each command in detail later in this chapter. Usually *data_ptr* points to a buffer that passes data between the user task and the driver and *n_bytes* defines the size of this buffer.

The argument *dev_info_ptr* is unused for the TPMC680 driver and should be set to NULL.

The following devctl command codes are defined in *tpmc680.h*:

Value	**Description**
DCMD_TP680_SET_MODE	Configure direction size and parameters of a specified port.
DCMD_TP680_GET_8BIT_PORT	Get input value of a specified 8-bit port.
DCMD_TP680_SET_8BIT_PORT	Set value of a specified 8-bit output port.
DCMD_TP680_READ_16BIT_DATA	Read data received on a specified 16-bit input port.
DCMD_TP680_WRITE_16BIT_DATA	Send data via a specified 16-bit output port.
DCMD_TP680_READ_32BIT_DATA	Read data received on the 32-bit input port.
DCMD_TP680_WRITE_32BIT_DATA	Send data via the 32-bit output port.
DCMD_TP680_GET_64BIT_PORT	Get input value of all 64 I/O lines.
DCMD_TP680_SET_64BIT_PORT	Set value to all 64 output lines.
DCMD_TP680_EVENT_WAIT	Wait for a specified event on a specified input line.

See behind for more detailed information on each control code.

To use these TPMC680 specific control codes the header file tpmc680.h must be included in the application.

RETURNS

On success, EOK is returned. In the case of an error, the appropriate error code is returned by the function (not in errno!).

ERRORS

Returns only Neutrino specific error codes, see Neutrino Library Reference.

Other function dependent error codes will be described for each devctl code separately.

The TPMC680 driver always returns standard QNX error codes.

SEE ALSO

Library Reference - devctl()

3.3.1 DCMD_TP680_SET_MODE

NAME

DCMD_TP680_SET_MODE – Configure port direction, size, and mode

DESCRIPTION

This function configures direction, size and mode of a specified port of the associated device. A pointer to the callers configuration buffer (*TP680_SET_MODE_BUF*) and the size of this structure are passed by the parameters *data_ptr* and *n_bytes* to the device.

The *TP680_SET_MODE_BUF* structure has the following layout:

```
typedef struct
{
      int               port;
      unsigned long     direction;
      unsigned long     mode;
      unsigned long     hsFlags;
} TP680_SET_MODE_BUF, *PTP680_SET_MODE_BUF;
```

port

This value specifies the port that should be configured. Valid port numbers are 0 up to 7. Some ports can not be configured to all modes.

direction

This value specifies the port direction. Dependent on the module some configurations are not allowed. The following values are defined:

define	**description**
TP680_IO_DIR_IN	configures the port as input port
TP680_IO_DIR_OUT	configures the port for output

mode

This value specifies the mode (width) of the port. Some mode changes will disconnect ports from the specified one, all disconnected ports will be set to 8-bit byte input mode. The following modes are predefined:

define	**description**
TP680_IO_MODE_BYTE	configures the port as an 8-bit byte I/O port
TP680_IO_MODE_HS16BIT	configures the port to work in 16-bit handshake mode
TP680_IO_MODE_HS32BIT	configures the port to work in 32-bit handshake mode
TP680_IO_MODE_SYNCHRON	configures the port to work in 64-bit synchronous byte I/O mode

hsFlags

This argument specifies the handshake mode and FIFO event. A handshake mode and a handshake event can be ORed.
The following values are defined for valid interrupt modes:

define	**description**
TP680_IO_HSFLAG_NO	No handshake output signal
TP680_IO_HSFLAG_INTERLOCKED	Output handshake is generated in interlocked mode
TP680_IO_HSFLAG_PULSED	Output handshake is generated in pulsed mode

The following values are defined for FIFO events:

define	**description**
TP680_IO_HSFIFOEV_NOTFULL	The FIFO event is generated if the FIFO is not filled.
TP680_IO_HSFIFOEV_EMPTY	The FIFO event is generated if the FIFO is empty.

Please refer to the User Manual of TPMC680 to understand all modes and dependencies.

EXAMPLE

```
int                     fd;
int                     result;
TP680_SET_MODE_BUF      modeBuf;

/* Configure Port 2 for 16-bit HS output */
/* - interlocked mode */
/* - FIFO event on buffer empty */
modeBuf.port = 2;
modeBuf.direction = TP680_IO_DIR_OUT;
modeBuf.mode = TP680_IO_MODE_HS16BIT;
modeBuf.hsFlags = TP680_IO_HSFLAG_INTERLOCKED | TP680_IO_HSFIFOEV_EMPTY;

result = devctl(   fd,
                   DCMD_TP680_SET_MODE,
                   &modeBuf,
                   sizeof(modeBuf),
                   NULL);
if (result != EOK)
{
     /* process devctl() error */
}
```

ERRORS

EINVAL	An argument specified value is invalid.
ECHRNG	The port number is out of range or the port number is not valid for the specified configuration.
EACCES	The port can not be accessed. It is connected to another port.

SEE ALSO

Library Reference - devctl()

3.3.2 DCMD_TP680_GET_8BIT_PORT

NAME

DCMD_TP680_GET_8BIT_PORT – Get value of an 8-bit port

DESCRIPTION

This function reads the actual value of a specified 8-bit port of the associated device. A pointer to a callers 8-bit buffer (*TP680_8BIT_PORT_BUF*) and the size of this structure are passed by the parameters *data_ptr* and *n_bytes* to the device.

The *TP680_8BIT_PORT_BUF* structure has the following layout:

```
typedef struct
{
      int             port;
      unsigned char   value;
} TP680_8BIT_PORT_BUF, *PTP680_8BIT_PORT_BUF;
```

port

This value specifies the port that should be read. Valid port numbers are 0 up to 7. The port must be in byte mode.

value

This is the parameter where the input value will be stored to.

The lower two bits of port 4 and port 5 are set to zero, if the associated pins are used for handshake.

EXAMPLE

```
int                     fd;
int                     result;
TP680_8BIT_PORT_BUF     byteBuf;

/* Read actual state of port 2 input */
byteBuf.port = 2;

result = devctl(   fd,
                   DCMD_TP680_GET_8BIT_PORT,
                   &byteBuf,
                   sizeof(byteBuf),
                   NULL);
if (result != EOK)
{
     /* process devctl() error */
}
printf(“INPUT: %02Xh\n”, byteBuf.value);
```

ERRORS

ECHRNG	The port number is out of range.
EACCES	The port can not be accessed. It is not configured for 8-bit byte access.

SEE ALSO

Library Reference - devctl()

3.3.3 DCMD_TP680_SET_8BIT_PORT

NAME

DCMD_TP680_SET_8BIT_PORT – Set value to an 8-bit output port

DESCRIPTION

This function sets the actual value of a specified 8-bit output port of the associated device. A pointer to a callers 8-bit buffer (*TP680_8BIT_PORT_BUF*) and the size of this structure are passed by the parameters *data_ptr* and *n_bytes* to the device.

The *TP680_8BIT_PORT_BUF* structure has the following layout:

```
typedef struct
{
        int                     port;
        unsigned char           value;
} TP680_8BIT_PORT_BUF, *PTP680_8BIT_PORT_BUF;
```

port

This value specifies the port that should be changed. Valid port numbers are 0 up to 7. The port must be in byte output mode.

value

This is the parameter specifies the new output value.

The lower two bits of port 4 and port 5 are ignored, if the associated pins are used for handshake.

EXAMPLE

```
int                     fd;
int                     result;
TP680_8BIT_PORT_BUF     byteBuf;

/* Set port 2 to 0x12 */
byteBuf.port = 2;
byteBuf.value = 0x12;

result = devctl(    fd,
                    DCMD_TP680_GET_8BIT_PORT,
                    &byteBuf,
                    sizeof(byteBuf),
                    NULL);
if (result != EOK)
{
    /* process devctl() error */
}
```

ERRORS

ECHRNG	The port number is out of range.
EACCES	The port can not be accessed. It is not configured for 8-bit byte access.

SEE ALSO

Library Reference - devctl()

3.3.4 DCMD_TP680_READ_16BIT_DATA

NAME

DCMD_TP680_READ_16BIT_DATA – Read data from 16-bit input port

DESCRIPTION

This function reads received data from a specified 16-bit input port of the associated device. A pointer to a callers 16-bit buffer (*TP680_16BIT_BUF*) and the size of this structure are passed by the parameters *data_ptr* and *n_bytes* to the device.

The data structure has a fixed size. It is configured with the definition of *TP680_MAX_16BIT_ELEM*. This definition sets the maximum number of 16-bit words that can be read with one call.

It is possible to fill up a partially filled buffer, by simply recalling *DCMD_TP680_READ_16BIT_DATA* with the same buffer, without changing any values. This may be necessary to get packets of a fixed length.

The function always returns data that are stored in the receive FIFO, it will not wait until data is available.

The *TP680_16BIT_BUF* structure has the following layout:

```
typedef struct
{
        int                 port;
        int                 maxElements;
        int                 usedElements;
        unsigned short      value[TP680_MAX_16BIT_ELEM];
} TP680_16BIT_BUF, *PTP680_16BIT_BUF;
```

port

This value specifies the port the data should be read from. Valid port numbers for this function are 0 and 2. The port must be in 16-bit handshake input mode.

maxElements

This value specifies the number of 16-bit values that will be filled into the buffer at maximum. This value can be different to the definition *TP680_MAX_16BIT_ELEM*, but must not be greater.

used Elements

This value returns the number of received 16-bit data words in the buffer. This value should be set 0 before calling the function, or it should keep the returned value, if the buffer should be filled up with the next call.

value[]

This is the buffer where the received data will be copied to.

EXAMPLE

```
int                  fd;
int                  result;
TP680_16BIT_BUF      inputBuf;

/* Read a block of 20 words from port 0, */
/* retry until 20 words are received */
inputBuf.port = 0;
inputBuf.maxElement = 20;
inputBuf.usedElements = 0;

do
{
     result = devctl(   fd,
                        DCMD_TP680_READ_16BIT_DATA,
                        &inputBuf,
                        sizeof(inputBuf),
                        NULL);
     if (result != EOK)
     {
          /* process devctl() error */
          break;
     }
} while (inputBuf.usedElements < 20);

/* 20 words of data received */
```

ERRORS

ECHRNG	The port number is out of range, the port number is not allowed.
EACCES	The port can not be accessed. It is not configured for 16-bit handshake input access.

SEE ALSO

Library Reference - devctl()

3.3.5 DCMD_TP680_WRITE_16BIT_DATA

NAME

DCMD_TP680_WRITE_16BIT_DATA – Write data to 16-bit output port

DESCRIPTION

This function writes data to a specified 16-bit output port of the associated device. A pointer to a callers 16-bit buffer (*TP680_16BIT_BUF*) and the size of this structure are passed by the parameters *data_ptr* and *n_bytes* to the device.

The data structure has a fixed size. It is configured with the definition of *TP680_MAX_16BIT_ELEM*. This definition sets the maximum number of 16-bit words that can be written with one call.

It is possible to retry sending a partially sent buffer, by recalling *DCMD_TP680_WRITE_16BIT_DATA* with the same buffer, without changing any values. This makes it easier to send packets of a fixed length.

The function copies the supplied data into drivers FIFO. It will not wait until data is sent. The function return immediately when all data is copied into the FIFO or the FIFO is filled.

The *TP680_16BIT_BUF* structure has the following layout:

```
typedef struct
{
      int                 port;
      int                 maxElements;
      int                 usedElements;
      unsigned short      value[TP680_MAX_16BIT_ELEM];
} TP680_16BIT_BUF, *PTP680_16BIT_BUF;
```

port

> This value specifies the port the data should be send to. Valid port numbers for this function are 0 and 2. The port must be in 16-bit handshake output mode.

maxElements

> This value specifies the number of 16-bit values that are stored into the buffer and shall be sent. This value can be different to the definition *TP680_MAX_16BIT_ELEM*, but must not be greater.

used Elements

> This value returns the number of received 16-bit data words that have been copied into the transmit FIFO. This value should be set 0 before calling the function or it should be left to the previous returned value if unsent data should be sent with a recall of the function.

value[]

> This is the data buffer that should be sent.

EXAMPLE

```
int                  fd;
int                  result;
TP680_16BIT_BUF      outputBuf;

/* Send a block of 20 words from port 2, */
/* retry until 20 words are written */
outputBuf.port = 2;
outputBuf.maxElement = 20;
outputBuf.usedElements = 0;
/* Fill up data */
outputBuf.value[0] = 0x1234;
…
outputBuf.value[19] = 0x4321;

do
{
     result = devctl(   fd,
                        DCMD_TP680_WRITE_16BIT_DATA,
                        &outputBuf,
                        sizeof(outputBuf),
                        NULL);
     if (result != EOK)
     {
          /* process devctl() error */
          break;
     }
} while (outputBuf.usedElements < 20);

/* 20 words of data sent */
```

ERRORS

ECHRNG The port number is out of range, the port number is not allowed.

EACCES The port can not be accessed. It is not configured for 16-bit handshake output access.

SEE ALSO

Library Reference - devctl()

3.3.6 DCMD_TP680_READ_32BIT_DATA

NAME

DCMD_TP680_READ_32BIT_DATA – Read data from 32-bit input port

DESCRIPTION

This function reads received data from the 32-bit input port of the associated device. A pointer to a callers 32-bit buffer (*TP680_32BIT_BUF*) and the size of this structure are passed by the parameters *data_ptr* and *n_bytes* to the device.

The data structure has a fixed size. It is configured with the definition of *TP680_MAX_32BIT_ELEM*. This definition sets the maximum number of 32-bit words that can be read with one call.

It is possible to fill up a partially filled buffer, by simply recalling *DCMD_TP680_READ_32BIT_DATA* with the same buffer, without changing any values. This may be necessary to get packets of a fixed length.

The function always returns data that are stored in the receive FIFO, it will not wait until data is available.

The *TP680_32BIT_BUF* structure has the following layout:

```
typedef struct
{
        int                     maxElements;
        int                     usedElements;
        unsigned short          value[TP680_MAX_32BIT_ELEM];
} TP680_32BIT_BUF, *PTP680_32BIT_BUF;
```

maxElements

This value specifies the number of 32-bit values that will be filled into the buffer at maximum. This value can be different to the definition *TP680_MAX_32BIT_ELEM*, but must not be greater.

used Elements

This value returns the number of received 32-bit data words in the buffer. This value should be set 0 before calling the function, or it should keep the returned value, if the buffer should be filled up with the next call.

value[]

This is the buffer where the received data will be copied to.

EXAMPLE

```
int                  fd;
int                  result;
TP680_32BIT_BUF      inputBuf;

/* Read a block of 20 longwords from 32-bit, */
/* retry until 20 longwords are received */
inputBuf.maxElement = 20;
inputBuf.usedElements = 0;

do
{
     result = devctl(   fd,
                        DCMD_TP680_READ_32BIT_DATA,
                        &inputBuf,
                        sizeof(inputBuf),
                        NULL);
     if (result != EOK)
     {
          /* process devctl() error */
          break;
     }
} while (inputBuf.usedElements < 20);

/* 20 longwords of data received */
```

ERRORS

EACCES	The port can not be accessed. It is not configured for 32-bit handshake input access.

SEE ALSO

Library Reference - devctl()

3.3.7 DCMD_TP680_WRITE_32BIT_DATA

NAME

DCMD_TP680_WRITE_32BIT_DATA – Write data to 32-bit output port

DESCRIPTION

This function writes data to the 32-bit output port of the associated device. A pointer to a callers 32-bit buffer (*TP680_32BIT_BUF*) and the size of this structure are passed by the parameters *data_ptr* and *n_bytes* to the device.

The data structure has a fixed size. It is configured with the definition of *TP680_MAX_32BIT_ELEM*. This definition sets the maximum number of 32-bit words that can be written with one call.

It is possible to retry sending a partially sent buffer, by recalling *DCMD_TP680_WRITE_32BIT_DATA* with the same buffer, without changing any values. This makes it easier to send packets of a fixed length.

The function copies the supplied data into drivers FIFO. It will not wait until data is sent. The function return immediately when all data is copied into the FIFO or the FIFO is filled.

The *TP680_32BIT_BUF* structure has the following layout:

```
typedef struct
{
        int                     maxElements;
        int                     usedElements;
        unsigned short          value[TP680_MAX_32BIT_ELEM];
} TP680_32BIT_BUF, *PTP680_32BIT_BUF;
```

maxElements

This value specifies the number of 32-bit values that are stored into the buffer and shall be sent. This value can be different to the definition *TP680_MAX_32BIT_ELEM*, but must not be greater.

used Elements

This value returns the number of received 32-bit data words that have been copied into the transmit FIFO. This value should be set 0 before calling the function or it should be left to the previous returned value if unsent data should be sent with a recall of the function.

value[]

This is the data buffer that should be sent.

EXAMPLE

```
int                 fd;
int                 result;
TP680_32BIT_BUF     outputBuf;

/* Send a block of 20 longwords, */
/* retry until 20 longwords are written */
outputBuf.maxElement = 20;
outputBuf.usedElements = 0;
/* Fill up data */
outputBuf.value[0] = 0x12345678;
…
outputBuf.value[19] = 0x87654321;

do
{
     result = devctl(   fd,
                        DCMD_TP680_WRITE_32BIT_DATA,
                        &outputBuf,
                        sizeof(outputBuf),
                        NULL);
     if (result != EOK)
     {
          /* process devctl() error */
          break;
     }
} while (outputBuf.usedElements < 20);

/* 20 longwords of data sent */
```

ERRORS

EACCES	The port can not be accessed. It is not configured for 32-bit handshake output access.

SEE ALSO

Library Reference - devctl()

3.3.8 DCMD_TP680_GET_64BIT_PORT

NAME

DCMD_TP680_GET_64BIT_PORT – Get values of all 64 I/O lines

DESCRIPTION

This function reads the actual value of all 64 I/O lines of the associated device. A pointer to a callers buffer (*TP680_64BIT_PORT_BUF*) and the size of this structure are passed by the parameters *data_ptr* and *n_bytes* to the device.

The *TP680_64BIT_PORT_BUF* structure has the following layout:

```
typedef struct
{
        unsigned long           value_31_0;
        unsigned long           value_63_32;
} TP680_64BIT_PORT_BUF, *PTP680_64BIT_PORT_BUF;
```

value_31_0

This argument returns the actual state of I/O line 0 up to 31.

value_63_32

This argument returns the actual state of I/O line 32 up to 63.

EXAMPLE

```
int                     fd;
int                     result;
TP680_64BIT_PORT_BUF    inBuf;

/* Read actual state the I/O lines */
result = devctl(    fd,
                    DCMD_TP680_GET_64BIT_PORT,
                    &inBuf,
                    sizeof(inBuf),
                    NULL);
if (result != EOK)
{
     /* process devctl() error */
}
printf(“INPUT: %08lX %08lX h\n”,
              byteBuf. value_64_32,
              byteBuf. value_31_0);
```

ERRORS

EACCES	The port can not be accessed. It is not configured for 64-bit byte mode.

SEE ALSO

Library Reference - devctl()

3.3.9 DCMD_TP680_SET_64BIT_PORT

NAME

DCMD_TP680_SET_64BIT_PORT – Set values of all 64 output lines

DESCRIPTION

This function sets all 64 output lines of the associated device. A pointer to a callers buffer (*TP680_64BIT_PORT_BUF*) and the size of this structure are passed by the parameters *data_ptr* and *n_bytes* to the device.

The *TP680_64BIT_PORT_BUF* structure has the following layout:

```
typedef struct
{
    unsigned long          value_31_0;
    unsigned long          value_63_32;
} TP680_64BIT_PORT_BUF, *PTP680_64BIT_PORT_BUF;
```

value_31_0

This argument specifies the new output value of output line 0 up to 31.

value_63_32

This argument specifies the new output value of output line 32 up to 63.

EXAMPLE

```
int                     fd;
int                     result;
TP680_64BIT_PORT_BUF    outBuf;

/* Set all output lines to 0 */
outBuf.value_31_0 = 0;
outBuf.value_63_32 = 0;
result = devctl(   fd,
                   DCMD_TP680_SET_64BIT_PORT,
                   &outBuf,
                   sizeof(outBuf),
                   NULL);
if (result != EOK)
{
    /* process devctl() error */
}
```

ERRORS

EACCES	The port can not be accessed. It is not configured for 64-bit output mode.

SEE ALSO

Library Reference - devctl()

3.3.10 DCMD_TP680_EVENT_WAIT

NAME

DCMD_TP680_EVENT_WAIT – Wait for a specified input transition

DESCRIPTION

This function waits for a input transition on a specified I/O line of the associated device. A pointer to a callers buffer (*TP680_EVENT_BUF*) and the size of this structure are passed by the parameters *data_ptr* and *n_bytes* to the device.

The *TP680_EVENT_BUF* structure has the following layout:

```
typedef struct
{
        int             eventLine;
        int             eventType;
        int             eventTimeout;
} TP680_EVENT_BUF, *PTP680_EVENT_BUF;
```

eventLine

This argument specifies the I/O the event should occur on. Valid values are 0 up to 63.

eventType

This argument specifies the transition type to wait for. The following types are defined in *tpmc680.h*:

define	**description**
TP680_EV_LO2HI_TRANS	Event occurs, if a low to high transition is detected.
TP680_EV_HI2LO_TRANS	Event occurs, if a high to low transition is detected
TP680_EV_ANY_TRANS	Event occurs, on every transition

eventTimeout

This parameter specifies the maximum time to wait for the specified event. If the specified time has occurred, the call will return with an error. The time is specified in seconds. A value of -1 means, that no timeout is used.

EXAMPLE

```
int                fd;
int                result;
TP680_EVENT_BUF    eventBuf;

/* Wait for a high to low transition on I/O line 17, */
/* timeout after 20 seconds */
eventBuf.eventLine = 17;
eventBuf.eventType = TP680_EV_LO2HI_TRANS;
eventBuf.eventTimeout = 20;
result = devctl(   fd,
                   DCMD_TP680_EVENT_WAIT,
                   &eventBuf,
                   sizeof(eventBuf),
                   NULL);
if (result != EOK)
{
     /* process devctl() error */
}
```

ERRORS

EACCES	The I/O line is configured for handshake mode and can not be used.
ECHRNG	The specified I/O line number is out of range, only 0 up to 63 is valid.
EINVAL	The transition type is invalid.
EBUSY	There is already a wait active on the specified I/O line.
ETIMEDOUT	The call has timed out.

SEE ALSO

Library Reference - devctl()