The Embedded I/O Company

TEWS TECHNOLOGIES

TPMC501-SW-95

QNX6 - Neutrino Device Driver

Optically Isolated 32 Channel 16 Bit ADC PMC

User Manual

Issue 1.1 Version 1.0.0

November 2003

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

TPMC501-SW-95

Optically Isolated 32 Channel 16 Bit ADC PMC

QNX6-Neutrino Device Driver

Issue	Description	Date
1.0	First Issue	July 25, 2003
1.1	Introduction corrected	November 10, 2003

Table of Content

1 Introduction

The TPMC501-SW-95 QNX6-Neutrino device driver allows the operation of a TPMC501 – Optically Isolated 32 Channel 16 Bit ADC PMC on QNX6-Neutrino operating systems with Intel or Intel-compatible x86 CPUs.

The TPMC550 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.

To start an I/O request the client process has to send an appropriate message, which contains a function code and optional parameter, to the server process. After the I/O operation has finished the server process replies the request message with a completion status and optional process data to caller.

The TPMC501 device driver includes the following functions:

- Configure attached module-type
- reading an analog input value (single channel)
- configuring, starting, stopping and using sequencer mode

2 Installation

The software is delivered on a PC formatted 3½" HD diskette, stored in a gzipped tar-archive named *TPMC501-SW-95.tar.gz*. This manual is located on the disk too, named *TPMC501-SW-95.pdf*.

Following files are stored in the tar-archive:

/driver/tpmc501.c	Driver source code
/driver/tpmc501.h	Definitions and data structures for driver and application
/driver/tpmc501def.h	Device driver include
/driver/node.c	Queue management source code
/driver/node.h	Queue management definitions
/example/example.c	Example application

For installation copy the tar-archive into the /usr/src directory and unpack it (e.g. `tar –xvzf TPMC501-SW-95.tar.gz`). After that the necessary directory structure for the automatic build and the source files are available underneath the new directory called *tpmc501*.

It is absolutely important to extract the TPMC501 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/tpmc501/driver directory

Execute the Makefile:

```
# make install
```

After successful completion the driver binary (tpmc501) will be installed in the /bin directory.

2.2 Build the example application

Change to the /usr/src/tpmc501/example directory

Execute the Makefile:

```
# make install
```

After successful completion the example binary (tp501exam) will be installed in the /bin directory.

2.3 Start the driver process

To start the TPMC501 device driver respective you have to enter the process name with optional parameter from the command shell or in the startup script.

```
tpmc501 [-v] &
```

The TPMC501 Resource Manager registers created devices in the Neutrinos pathname space under following names.

```
/dev/tpmc501_0
/dev/tpmc501_1
…
/dev/tpmc501_x
```

This pathname must be used in the application program to open a path to the desired TPMC501 device.

```
fd = open(“/dev/tpmc501_0”, O_RDWR);
```

For debugging you can start the TPMC501 Resource Manager with the –v option. Now the Resource Manager will print versatile information about TPMC501 configuration and command execution on the terminal window.

```
tpmc501 –v &
```

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 TPMC501 named by pathname. The flags argument controls how the file is to be opened. TPMC501 devices must be opened O_RDWR.

EXAMPLE

```
int fd;

fd = open(“/dev/tpmc501_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 TPMC501 driver and should be set to NULL.

The following devctl command codes are defined in *tpmc501.h*:

Value	Description
DCMD_TPMC501_READ	Read a new ADC input value
DCMD_TPMC501_CONFIG	Configure the module type
DCMD_TPMC501_SEQ_CONF	Configure channel for sequencer mode
DCMD_TPMC501_SEQ_START	Start sequencer mode
DCMD_TPMC501_SEQ_STOP	Stop sequencer mode

See behind for more detailed information on each control code.

To use these TPMC501 specific control codes the header file TPMC501.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 code, see Neutrino Library Reference.

Other function dependent error codes will be described for each devoctl code separately. Note, the TPMC501 driver always returns standard QNX error codes.

SEE ALSO

Library Reference - devctl()

3.3.1 DCMD_TPMC501_READ

NAME

DCMD_TPMC501_READ – Read a new ADC input value

DESCRIPTION

This function reads a new ADC input value from a specified channel and returns after conversion of the value to the caller. For this function a *TPMC501_IO_READ_STRUCT* structure must be initialized with appropriate data.

```
typedef struct
{
    int             channel;    // channel number
    unsigned short  gain;       // gain stage
    long            value;      // new output value
    long            flags;      // output flags
} TPMC510_IO_READ_STRUCT;
```

channel

This argument specifies the channel number. Valid channel numbers are 1 up to 32 for single-ended operation mode. For differential mode channel numbers from 1 to 16 are supported.

gain

This argument specifies the used gain level. There are four different gain levels available, the used gain factor depends on the attached TPMC501-module-type (e.g. for a TPMC501-10 module gain level 0 results in a gain factor 1, level 2 results in a factor 5). Valid gain levels are 0 up to 3.

value

In this argument the new input value will be stored. The input value range depends on the module type and the used gain level (e.g. the use of a TPMC501-10 module with a gain level of 1 (gain factor 2) results in an input-range of ±5V).

flags

This value is an ORed value of the following flags:

Value	Description
TP501_CORR_ENA	If this flag is set, the input value will be corrected with the factory stored correction data.
TP501_DIFF_MODE	If this flag is set, the differential conversion mode is used for data aquisition.

EXAMPLE

```
int                         fd;
int                         result;
TPMC501_IO_READ_STRUCT      ioBuf;

...

/*
** read value from channel 1 with correction
*/
ioBuf.channel = 1;
ioBuf.flags   = TP501_CORR_ENA;

result = devctl(   fd,
                   DCMD_TPMC501_READ,
                   &ioBuf,
                   sizeof(ioBuf),
                   NULL);
if (result == EOK)
{
     /* command successful */
     printf("value = %d \n", iobuf.value);
}

...
```

ERRORS

ECHRNG	Specified channel not supported by attached module.
EINVAL	Invalid gain value specified.
EBUSY	The channel can not be used, because the module is configured in sequencer mode.
ETIME	The module does not complete the conversion cycle within a normal time.

SEE ALSO

Library Reference - devctl()

3.3.2 DCMD_TPMC501_CONFIG

NAME

DCMD_TPMC501_ CONFIG – Configure the module type

DESCRIPTION

This function is used to tell the driver what type of a TPMC501 module is attached. Therefore a TPMC501_CONFIG_STRUCT structure must be filled with the correct value. The module type must be known for correct calculation of the converted ADC values.

typedef struct
{
    int type; // module type
} TPMC501_ CONFIG_STRUCT;

type

This value specifies the attached TPMC501-module-type. For valid configuration values see the header file *tpmc501.h*.

EXAMPLE

```
int                         fd;
int                         result;
TPMC501_CONFIG_STRUCT       cfgBuf;

...

/*
** configure module type
*/
cfgBuf.type = TPMC501_X1;        // e.g. use a TPMC501-11 module

result = devctl(   fd,
                   DCMD_TPMC501_CONFIG,
                   &cfgBuf,
                   sizeof(cfgBuf),
                   NULL);
if (result == EOK)
{
     /* successful read */
}

...
```

ERRORS

EINVAL	Invalid argument. This error code is returned if the specified model-type is invalid.

SEE ALSO

Library Reference - devctl()

3.3.3 DCMD_TPMC501_SEQ_CONF

NAME

DCMD_TPMC501_SEQ_CONF – Configures a channel for sequencer mode

DESCRIPTION

This function sets up a channel for use with sequencer mode. For this function the structure *TPMC501_IN_SEQCONF_STRUCT* must be initialized with appropriate data.

```
typedef struct
{
    int             channel;    // channel to be configured
    unsigned short  gain;       // gain level
    unsigned short  flags;      // special flags
} TPMC501_IN_SEQCONF_STRUCT;
```

channel

This argument specifies the channel to be configured. Valid values are 1 to 16/32 depending on the operation mode (differential or single-ended).

gain

This argument specifies the gain level to be used. There are four different gain levels available, the used gain factor depends on the attached TPMC501-module-type (e.g. for a TPMC501-10 module gain level 0 results in a gain factor 1, level 2 results in a factor 5). Valid gain levels are 0 up to 3.

flags

This value is an ORed value of the following flags:

Value	Description
TP501_CORR_ENA	If this flag is set, the input value will be corrected with the factory stored correction data.
TP501_DIFF_MODE	If this flag is set, the differential conversion mode is used for data acquisition.
TP501_CHAN_ENA	If this flag is set, the specified channels is enabled for sequencer mode. Otherwise it will not be converted during sequencer mode.

EXAMPLE

```
int                          fd;
int                          result;
TPMC501_IN_SEQCONF_STRUCT    seqCfgBuf;

...

/*
** configure channel 1 for sequencer in single-ended mode, use correction
*/
seqCfgBuf.channel = 1;
seqCfgBuf.gain    = 0;
seqCfgBuf.flags   = TP501_CHAN_ENA | TP501_CORR_ENA;

result = devctl(   fd,
                   DCMD_TPMC501_SEQ_CONF,
                   &seqCfgBuf,
                   sizeof(seqCfgBuf),
                   NULL);
if (result == EOK)
{
     /* sequencer mode configured */
}

...
```

ERRORS

ECHRNG	Specified channel not supported by attached module.
EINVAL	Invalid gain value specified.

SEE ALSO

Library Reference - devctl()

3.3.4 DCMD_TPMC501_SEQ_START

NAME

DCMD_TPMC501_SEQ_START – Start sequencer mode

DESCRIPTION

This function starts the sequencer mode and returns immediately to the caller. The sequencer starts to write its acquired data into a shared memory object defined and opened by the user-application. The data is transferred between the driver and the calling application via the ringbuffer structure defined by TPMC501_RINGBUF. To start the sequencer the structure TPMC501_IN_SEQSTART_STRUCT must be filled with appropriate data.

```
typedef struct
{
    unsigned short      seqTime;        // time-factor for sequencer timer
    char                shMemName[15];  // name of the shared memory object
} TPMC501_IN_SEQSTART_STRUCT;
```

seqTime

This argument specifies the timer factor used by the sequencer to convert the values from the specified channels. The built-in timer uses a 1/10 ms timeslot, so fixed frequencies up to 10kHz can be specified. If *seqTime* is 0, the sequencer works in runaround-mode, that means after the last conversion of the sequence it starts immediately from the beginning. The sampling frequency depends on the number of channels to be converted.

shMemName[15]

This argument specifies the unique name of the needed shared memory object. It has been limited to 15 characters.

```
typedef struct
{
    int                 channel[32];    // value array for each channel
} TPMC501_SEQ_ENTRY;

typedef struct
{
    TPMC501_SEQ_ENTRY buffer[TP501_RINGBUF_SIZE];  // ringbuffer for acquired data
    unsigned long       putPtr, getPtr;  // positions for read and write access to the buffer
    char                status;          // status of the sequencer
} TPMC501_RINGBUF;
```

buffer[]

This argument specifies the data exchange ringbuffer for data transfer. Only the driver process should write to this memory section. Every buffer entry contains a data storage for one complete sequence (32 values). To change the ringbuffer size, edit the value of TP501_RINGBUF_SIZE defined in the header file *tpmc501.h*.

The driver and the example must be recompiled and restarted after changing *TP501_RINGBUF_SIZE*.

putPtr

This argument specifies the position where the driver will fill in the next acquired value. After the new sequence values were written to the buffer, *putPtr* will be set to the next location.

getPtr

This argument specifies the position where the application can read a new value. After the read-operation the application has to increase the *getPtr*-value to get to the next read-position. It is necessary to use this value out of the shared memory object because the driver has to know the current reading position to avoid overwriting data.

status

This argument describes the actual status of the sequencer. Possible values are as follows:

Value	Description
TP501_SEQ_OK	Sequencer is working fine, no errors.
TP501_SEQ_ERR_TIMER	The specified timer value is too small for all channels to finish conversion.
TP501_SEQ_ERR_SW_DATA_OF	The ringbuffer is full, the driver cannot write new values, data is lost. The reading from the buffer was too slow.
TP501_SEQ_ERR_HW_DATA_OF	The TPMC501-module signals a hardware buffer overflow. The driver has not fetched the new values from the sequencer ram.

EXAMPLE

```
int                          fd, sharedmemfd;
int                          result;
TPMC501_IN_SEQSTART_STRUCT   seqStartBuf;
TPMC501_RINGBUF              *pRingBuf;

...

/*
** initialize shared memory object. don't forget to handle errors!
*/
sharedmemfd = shm_open(SHARED_MEMORY_NAME, O_RDWR | O_CREAT, 0777);
ftruncate(sharedmemfd, sizeof(TPMC501_RINGBUF));
pRingBuf = mmap( 0, sizeof(TPMC501_RINGBUF),
                 PROT_READ | PROT_WRITE, MAP_SHARED, sharedmemfd, 0);
```

```
/*
** start sequencer mode with a 10 * 1/10 ms timer factor
*/
seqStartBuf.seqTime   = 10;
seqStartBuf.shMemName = “/ringbuffer”;

result = devctl(   fd,
                   DCMD_TPMC501_SEQ_START,
                   seqStartBuf,
                   sizeof(seqStartBuf),
                   NULL);
if (result == EOK)
{
     /* sequencer successfully started */

     /** see example application and “Programming Hints” for read method */
}

...
```

ERRORS

ENOBUFS	Open of shared memory object failed.
ENOSPC	Setting the size for the shared memory object failed.
EACCES	Mapping of the shared memory object failed.
ETIME	Sequencer refused to start within specific timeout.

SEE ALSO

Library Reference - devctl()

3.3.5 DCMD_TPMC501_SEQ_STOP

NAME

DCMD_TPMC501_SEQ_STOP – Stop sequencer mode

DESCRIPTION

This function stops the sequencer mode, no option have to be supplied. It returns immediately to the caller.

EXAMPLE

```
int                         fd;
int                         result;

...

/*
**  stop sequencer mode
*/

result = devctl(   fd,
                   DCMD_TPMC501_ SEQ_STOP,
                   NULL,
                   0,
                   NULL);
if (result == EOK)
{
     /* sequencer stop successful */
}

...
```

ERRORS

ETIME	The sequencer refused to stop within specific timeout.

SEE ALSO

Library Reference - devctl()

4 Programming Hints

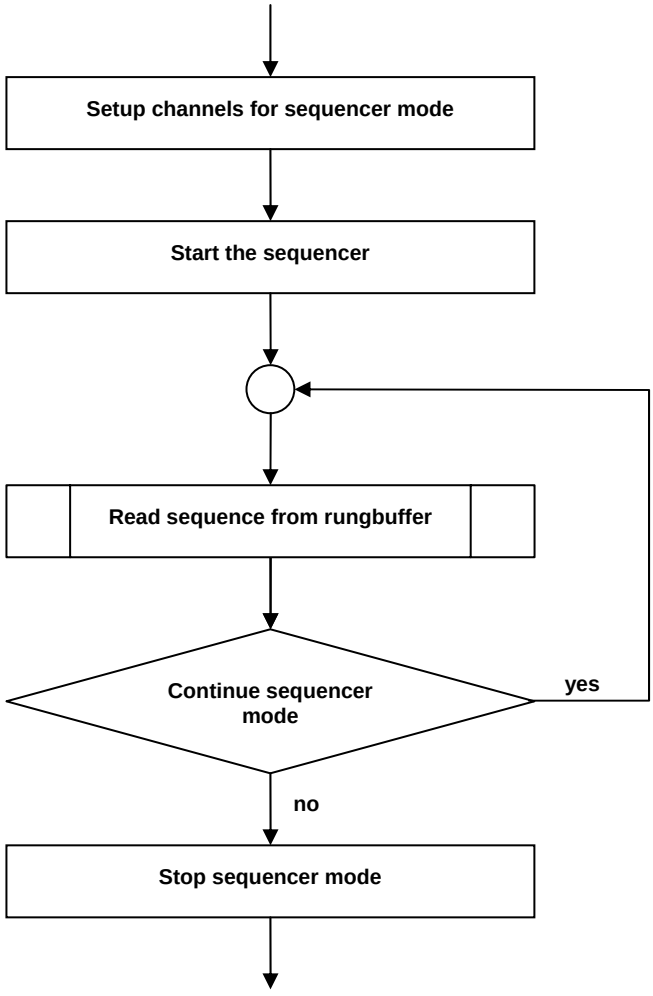
4.1 Using the sequencer mode

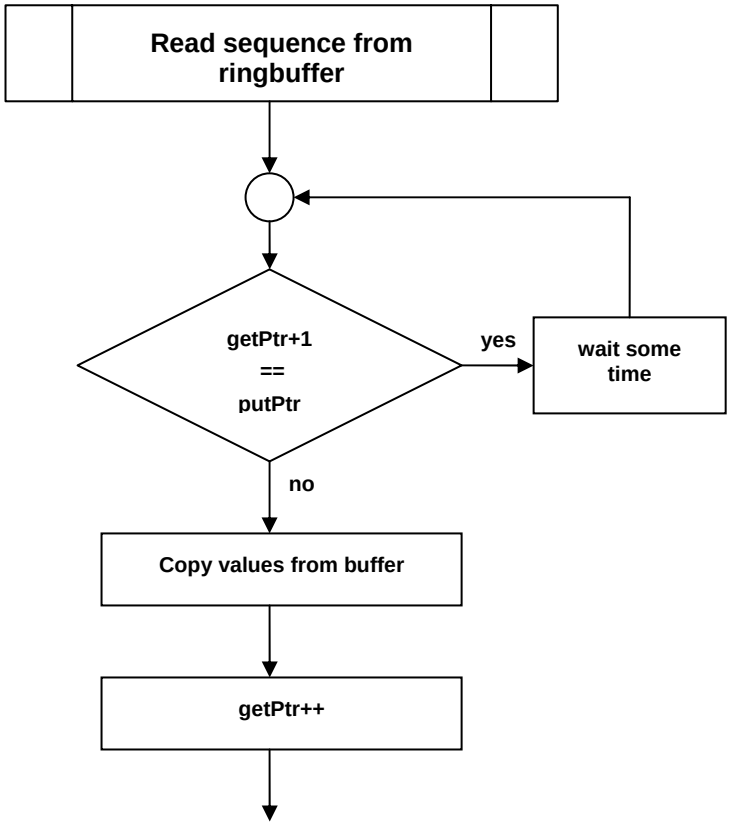
The following flow-charts are describing the high-performance sequencer mode. Because of the shared-memory usage no task-switching overhead slows down the data acquisition and transfer from the driver to the application.

Depending on the interrupt situation in the system, the number of channels to be used and the ringbuffer size sampling rates of up to 50kHz could be reached.

Make sure that the application reads the data fast enough. Do not try to print the acquired data on the screen via *printf* in a high frequency application. Use a file stream instead for debugging.

Note that for every completed sequencer cycle an interrupt is generated by the TPMC501 handled by the driver!




Read sequence from ringbuffer
getPtr+1 == putPtr
yes
wait some time
no
Copy values from buffer
getPtr++