

TPMC551-SW-82

Linux Device Driver

8 Channel 16 Bit DAC

Version 1.2.x

User Manual

Issue 1.2.0

March 2005

TEWS TECHNOLOGIES GmbH

Am Bahnhof 7
Phone: +49-(0)4101-4058-0
e-mail: info@tews.com

25469 Halstenbek / Germany
Fax: +49-(0)4101-4058-19
www.tews.com

TEWS TECHNOLOGIES LLC

1 E. Liberty Street, Sixth Floor
Phone: +1 (775) 686 6077
e-mail: usasales@tews.com

Reno, Nevada 89504 / USA
Fax: +1 (775) 686 6024
www.tews.com

TPMC551-SW-82

8 Channel 16 Bit DAC

Linux 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.

©2001-2005 by TEWS TECHNOLOGIES GmbH

Issue	Description	Date
1.0	First Issue	April 22, 2002
1.1	Parameter for TP551_IOCGRADPARAM changed	May 13, 2002
1.2.0	Kernel 2.6.x Support	March 21, 2005

Table of Contents

1	INTRODUCTION.....	4
2	INSTALLATION.....	5
	2.1 Build and install the device driver.....	5
	2.2 Uninstall the device driver	5
	2.3 Install device driver into the running kernel	5
	2.4 Remove device driver from the running kernel	6
	2.5 Change Major Device Number	7
3	DEVICE INPUT/OUTPUT FUNCTIONS	8
	3.1 open()	8
	3.2 close().....	10
	3.3 write()	11
	3.4 ioctl()	13
	3.4.1 TP551_IOCGRADPARAM	15
	3.4.2 TP551_IOCSTOPSEQ	17
	3.4.3 TP551_IOCSTARTSEQ	18
	3.4.4 TP551_IOCWRITESEQ.....	20
4	DIAGNOSTIC.....	22

1 Introduction

The TPMC551-SW-82 Linux device driver allows the operation of a TPMC551 on Linux operating systems.

The TPMC551 device driver includes the following features:

- write a new value to a selected DAC channel
- use sequencer mode for continuously write to selected channels
- correction of input values with the factory programmed correction data

In case of difficulties during driver installation please contact TEWS TECHNOLOGIES.

2 Installation

The directory TPMC551-SW-82 on the distribution media contains the following files:

TPMC551-SW-82.pdf	This manual in PDF format
TPMC551-SW-82-SRC.tar.gz	GZIP compressed archive with driver source code
Release.txt	Release information

The GZIP compressed archive TPMC551-SW-82-SRC.tar.gz contains the following files and directories:

tpmc551.c	Driver source code
tpmc551def.h	Driver private include file
tpmc551.h	Driver public include file for application program
Makefile	Device driver make file
Makefile.elinos	Device driver make file for ELinOS
makenode	Script to create device nodes on the file system
tpxxhwdep.c	Low level WINNT style hardware access functions source file
tpxxhwdep.h	Access functions header file
tpmodule.c	Driver independent library
tpmodule.h	Driver independent library header file
Example/tpmc551example.c	Example application
Example/Makefile	Example application make file

In order to perform an installation, extract all files of the archive TPMC551-SW-82-SRC.tar.gz to the desired target directory.

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
```

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 tpmc551drv
```

- After the first build or if you are using dynamic major device allocation it's necessary to create new device nodes on the file system. Please execute the script file *makenode* to do this. If your kernel has enabled the device file system (devfs) 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 TPMC551 module found. The first TPMC551 module can be accessed with device node `/dev/tpmc551_0`, the second with device node `/dev/tpmc551_1` and so on.

The assignment of device nodes to physical TPMC551 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 tpmc551drv

If your kernel has enabled devfs, all `/dev/tpmc551_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 "*tpmc551drv: 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 DEVFS installed. The TPM500 driver use dynamic allocation of major device numbers per default. If this isn't suitable for the application it's possible to define a major number for the driver.

To change the major number edit the file `tpmc551def.h`, change the following symbol to appropriate value and enter `make install` to create a new driver.

`TPMC551_MAJOR` Valid numbers are in range between 0 and 255. A value of 0 means dynamic number allocation.

Example:

```
#define TPMC551_MAJOR 122
```

3 Device Input/Output functions

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

3.1 open()

NAME

open() opens a file descriptor.

SYNOPSIS

```
#include <fcntl.h>
```

```
int open (const char *filename, int flags)
```

DESCRIPTION

The **open** function creates and returns a new file descriptor for the file named by *filename*. The *flags* argument controls how the file has to be opened. This is a bit mask. Create the value by the bitwise OR of the appropriate parameters (using the | operator in C). See also the GNU C Library documentation for more information about the open function and open flags.

EXAMPLE

```
int hCurrent;  
  
...  
hCurrent = open("/dev/tpmc551_0", O_RDWR);  
...
```

RETURNS

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

ERRORS

`E_NODEV` The requested minor device does not exist.

This is the only error code returned by the driver, other codes may be returned by the I/O system during open. For more information about open error codes, see the *GNU C Library description – Low-Level Input/Output*.

SEE ALSO

GNU C Library description – Low-Level Input/Output

3.2 close()

NAME

close() closes a file descriptor.

SYNOPSIS

```
#include <unistd.h>
```

```
int close (int filedes)
```

DESCRIPTION

The **close** function closes the file descriptor *filedes*.

EXAMPLE

```
int hCurrent;  
...  
if (close(hCurrent) != 0)  
{  
    * handle close error conditions */  
}
```

RETURNS

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

ERRORS

E_NODEV The requested minor device does not exist.

This is the only error code returned by the driver, other codes may be returned by the I/O system during **close**. For more information about close error codes, see the *GNU C Library description – Low-Level Input/Output*.

SEE ALSO

GNU C Library description – Low-Level Input/Output

3.3 write()

NAME

write() – write to a device

SYNOPSIS

```
#include <unistd.h>
```

```
ssize_t write(int filedes, void *buffer, size_t size)
```

DESCRIPTION

The write function writes a DAC value to the specified channel.

A pointer to the callers write buffer (*TP551_WRITEBUF*) and the size of this structure is passed by the parameters *buffer* and *size* to the device.

The *TP551_WRITEBUF* structure has the following layout:

```
typedef struct
{
    unsigned short    channel;    /* channel number */
    unsigned short    flags;
    long              value;     /* ADC input value */
} TP551_WRITEBUF, *PTP551_WRITEBUF;
```

channel

This value specifies the DAC channel that will be used. Allowed values are 1 to 8 for TPMC551-10/-20 and 1 to 4 for TPMC551-11/-21.

flags

This value is an ORed value of the flags shown in the following table.

Name	Meaning
TP551_FL_CORR	If this flag is set, the driver will correct the DAC output value with the factory programmed correction data. If this flag is not set, the output value will not be corrected.
TP551_FL_LATCHED	If this flag is set the data will be loaded into the DAC, but the conversion will not be started, until the <i>TP551_FL_SIMCONV</i> flag is set.
TP551_FL_SIMCONV	This flag starts a simultaneous conversion for all channels. This flag is necessary to start a conversion in latched mode.

value

This parameter specifies the DAC output value. The value must be between 0 and 65535 for 0V..+10V mode and between -32768 and +32767 for -10V..+10V mode.

EXAMPLE

```
int hCurrent;
ssize_tNumBytes;
TP551_WRITEBUF DACBuf;
...
/*****'*****
Write channel 5 with corrected the input data
*****/
DACBuf.channel      = 5;
DACBuf.value        = 0x1234;
DACBuf.flags        = TP551_FL_CORR;

NumBytes = write(hCurrent, &DACBuf, sizeof(DACBuf));
if (NumBytes >= 0)
{
    printf( "\nWrite successful \n");
}
else
{
    printf("\nWrite failed --> Error = %d\n", errno );
}
...
```

RETURNS

On success write returns a positive value. In the case of an error, a value of -1 is returned. The global variable *errno* contains the detailed error code.

ERRORS

EINVAL	Invalid argument. This error code is also returned if the size of the write buffer is too small.
EFAULT	Invalid pointer to the write buffer.
EBUSY	The sequencer mode is active on the specified device.
ETIME	The settling or conversion exceeds the supposed range.

SEE ALSO

GNU C Library description – Low-Level Input/Output

3.4 ioctl()

NAME

ioctl() – device control functions

SYNOPSIS

```
#include <sys/ioctl.h>
```

```
int ioctl(int fildes, int request [, void *argp])
```

DESCRIPTION

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

The argument *request* specifies the control code for the operation. The optional argument *argp* depends on the selected request and is described for each request in detail later in this chapter.

The following ioctl codes are defined in *TPMC551*:

Value	Meaning
<i>TP551_IOCGREADPARAM</i>	Get module parameters including the factory programmed correction values, number of channels, voltage range selection.
<i>TP551_IOCSTOPSEQ</i>	Stop the sequencer
<i>TP551_IOCSTARTSEQ</i>	Setup and start the sequencer
<i>TP551_IOCWRITESEQ</i>	Write DAC data into sequencer FIFO-buffer

See below for more detailed information on each control code.

Note: To use these TPMC551 specific control codes the header file *tpmc551.h* must be included in the application.

RETURNS

On success, zero is returned. In the case of an error, a value of -1 is returned. The global variable *errno* contains the detailed error code.

ERRORS

EINVAL

Invalid argument. This error code is also returned if the requested ioctl function is unknown. Please check the argument *request*.

Other function dependant error codes will be described for each ioctl code separately. Note, the TPMC551 driver always returns standard Linux error codes.

SEE ALSO

ioctl man pages

3.4.1 TP551_IOCGRADPARAM

NAME

TP551_IOCGRADPARAM - Get the module parameters

DESCRIPTION

This ioctl function returns modules parameters, this includes the factory programmed correction data, number of channels and the voltage range selection.

A pointer to the callers parameter buffer (*TP551_PARABUF*) is passed by the parameter *argp* to the driver.

The *TP551_PARABUF* structure has the following layout:

```
typedef struct
{
    int                NumChan;    /* Number of Channels */
    int                biPol_1_4;  /* Voltagemode channel 1-4 */
    int                biPol_5_8;  /* Voltagemode channel 5-8 */
    signed short       OffsCorr[8]; /* Offset correction Data */
    signed short       GainCorr[8]; /* Gain correction Data */
} TP551_PARABUF, *PTP551_PARABUF;
```

NumChan

This parameter returns the number of DAC channels supported by the module.

biPol_1_4

This parameter returns TRUE, if the channels 1 to 4 are configured for -10V..+10V mode, if FALSE is returned, the channels are configured for 0V..+10V mode.

biPol_5_8

This parameter returns TRUE, if the channels 5 to 8 are configured for -10V..+10V mode, if FALSE is returned, the channels are configured for 0V..+10V mode.

OffsCorr

This array returns the factory programmed offset correction data set, that is used if the *TP551_FL_CORR* flag is set. The index of the array specifies the channel number, 0 selects channel 1, 1 selects channel 2 and so on.

GainCorr

This array returns the factory programmed gain correction data set, that is used if the *TP551_FL_CORR* flag is set. The index of the array specifies the channel number, 0 selects channel 1, 1 selects channel 2 and so on.

EXAMPLE

```
int hCurrent;
int result;
int x;
TP551_PARABUF ParamBuf;

...

result = ioctl(hCurrent, TP551_IOCTL_READPARAM, &ParamBuf);
if (result >= 0)
{
    for (x = 0; ParamBuf.NumChan < 8; x++)
        printf("Offset Error [%d] = %d \n",
            x + 1,
            ParamBuf.OffsCorr[x]);
    printf("Gain Error [%d] = %d \n",
        x + 1,
        ParamBuf.OffsCorr[x]);
}
else
{
    printf("\nRead module parameter failed --> Error = %d\n",
        errno);
}

...
```

ERRORS

EFAULT	Invalid pointer to the parameter buffer. Please check the argument <i>argp</i> .
--------	--

SEE ALSO

ioctl man pages

3.4.2 TP551_IOCSTOPSEQ

NAME

TP551_IOCSTOPSEQ – Stop Sequencer Mode

DESCRIPTION

This ioctl function stops the sequencer mode.

EXAMPLE

```
int hCurrent;
int result;

...

result = ioctl(hCurrent, TP551_IOCSTOPSEQ);
if (result >= 0)
{
    printf("\nStopping sequencer successful\n");
}
else
{
    printf("\nStopping sequencer failed --> Error = %d\n",
        errno);
}

...
```

ERRORS

This ioctl function returns no function specific error codes.

SEE ALSO

ioctl man pages

3.4.3 TP551_IOCSTARTSEQ

NAME

TP551_IOCSTARTSEQ - Setup and start the sequencer, enter sequencer mode

DESCRIPTION

This ioctl function sets up the TPMC551 to work in sequencer mode. The cycle time and the channel configuration are set up.

A pointer to the callers parameter buffer (*TP551_STARTSEQBUF*) is passed by the parameter *argp* to the driver.

The *TP551_STARTSEQBUF* structure has the following layout:

```
typedef struct
{
    unsigned short    channels;    /* channel number */
    unsigned short    cycleTime;  /* cycle time */
    unsigned short    flags;      /* flags */
} TP551_STARTSEQBUF, *PTP551_STARTSEQBUF;
```

channels

This parameter specifies which channel will be used in sequencer mode. Setting bit 0 will enable channel 1, setting bit 1 will enable channel 2 and so on.

cycleTime

This parameter specifies the cycle time that will be used. The value will be copied into the sequencer timer register. The value has a resolution of 100µs steps. If the flag ... is set the parameter will be ignored.

flags

The **flags** parameter is an ORed value of the following described flags.

Name	Meaning
TP551_FL_LATCHED	If this flag is set, the driver will output the data in latched mode, the output of all channels will be visible at the same time. Otherwise the data will be used in transparent mode.
TP551_FL_CONTINUOUS	The sequencer will work in continuous mode, data will be written as fast as possible to the output.

EXAMPLE

```

int hCurrent;
int result;
TP551_STARTSEQBUF SeqStartBuf;

...

/*****
Start sequencer with a cycle time of 1 sec
Enable following channels:
    Channel 1
    Channel 6
Use latched mode
*****/
SeqStartBuf.cycleTime = 10000;    /* 10000 * 100µs */
SeqStartBuf.channels = (1 << 0) | (1 << 5);    /* Enable channel */
SeqStartBuf.flags = TP551_FL_LATCHED;
result = ioctl(hCurrent, TP551_IOCSTARTSEQ, & SeqStartBuf);
if (result >= 0)
{
    printf("\nStarting sequencer successful\n");
}
else
{
    printf("\nStarting sequencer failed --> Error = %d\n",
        errno);
}

...

```

ERRORS

EFAULT Invalid pointer to the parameter buffer. Please check the argument *argp*.

SEE ALSO

ioctl man pages

3.4.4 TP551_IOCWRITESEQ

NAME

TP551_IOCWRITESEQ - Setup and start the sequencer, enter sequencer mode

DESCRIPTION

This ioctl function writes data into the sequencer FIFO. These data will be used by the interrupt function in sequencer mode to update the DAC output values.

A pointer to the callers parameter buffer (*TP551_WRITESEQBUF*) is passed by the parameter *argp* to the driver.

The *TP551_WRITESEQBUF* structure has the following layout:

```
typedef struct
{
    unsigned short    channels;    /* channel flags */
    unsigned short    correction; /* correction flags */
    unsigned short    values[8];  /* buffer */
    unsigned long     stat;       /* write status */
} TP551_WRITESEQBUF, *PTP551_WRITESEQBUF;
```

channels

This parameter specifies which channel shall update output data. Setting bit 0 will update channel 1, setting bit 1 will update channel 2 and so on. Channel which are activated and not specified to be updated, will held their value.

correction

This parameter specifies which channels shall the factory stored correction data. Setting bit 0 will enable data correction for channel 1, setting bit 1 will enable data correction for channel 2 and so on.

values

This array specifies the new output values. Setting bit 0 will enable data correction for channel 1, setting bit 1 will enable data correction for channel 2 and so on. The array index specifies the channel number the data assigned to. Index 0 for channel 1, index 1 for channel 2 and so on. The values must be between 0 and 65535 for 0V..+10V mode and between -32768 and +32767 for -10V..+10V mode. Only the values for channels specified for update will be used.

stat

This parameter returns the sequencer status. The status returns number of cycles which had not been used for new data output, because there has been no output data available in the FIFO. And the status can signal, that an output error has occurred. This will happen if the software is not able to handle a cycle before the next cycle starts. The stat argument is split in this way:

bits 27 .. 0	number of lost cycles
bit 30 (TP550_E_ERROR)	sequencer error has occurred

EXAMPLE

```
int hCurrent;
int result;
TP551_WRITESEQBUF SeqWriteBuf;

...
/*****
Update Sequencer data
Enable following channels:
    Channel 1
    Channel 6
Use correction for channel 6
*****/
SeqWriteBuf.channels = (1 << 0) | (1 << 5);
SeqWriteBuf.correction = (1 << 5);
SeqWriteBuf.values[0] = 0x1234;
SeqWriteBuf.values[5] = 0x7000;
result = ioctl(hCurrent, TP551_IOCWRITESEQ, & SeqWriteBuf);
if (result >= 0)
{
    printf("\nWriting sequencer data successful\n");
}
else
{
    printf("\nWriting sequencer data failed --> Error = %d\n",
        errno);
}
...

```

ERRORS

EFAULT Invalid pointer to the parameter buffer. Please check the argument *argp*.

SEE ALSO

ioctl man pages

4 Diagnostic

If the TPMC551 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 an correct running TPMC551 driver (see also the *proc* man pages).

```
# cat /proc/pci
...
Bus 0, device 9, function 0:
  Class 1180: PCI device 10b5:9050 (rev 1).
  IRQ 12.
  Non-prefetchable 32 bit memory at 0xe7000000 [0xe700007f].
  I/O at 0xe000 [0xe07f].
  I/O at 0xd800 [0xd81f].
  Non-prefetchable 32 bit memory at 0xe6800000 [0xe680003f].
...

# cat /proc/devices
Character devices:
  1 mem
  2 pty
  3 tty
  4 ttyS
  5 cua
  7 vcs
 10 misc
 29 fb
128 ptm
136 pts
162 raw
254 tpmc551drv
...

# cat /proc/interrupts
          CPU0
0:      1414319      XT-PIC  timer
1:        1535      XT-PIC  keyboard
2:           0      XT-PIC  cascade
4:         617      XT-PIC  serial
8:           2      XT-PIC  rtc
10:         14      XT-PIC  ncr53c8xx
11:        6758      XT-PIC  eth0
12:       16726      XT-PIC  TPMC551
14:       10362      XT-PIC  ide0
15:           5      XT-PIC  ide1
NMI:           0
```

```
ERR:          0
MIS:          0
```

```
# cat /proc/ioports
0000-001f : dma1
0020-003f : pic1
0040-005f : timer
0060-006f : keyboard
0070-007f : rtc
0080-008f : dma page reg
00a0-00bf : pic2
00c0-00df : dma2
00f0-00ff : fpu
0170-0177 : ide1
01f0-01f7 : ide0
02f8-02ff : serial(auto)
0376-0376 : ide1
03c0-03df : vga+
03f6-03f6 : ide0
03f8-03ff : serial(auto)
0cf8-0cff : PCI conf1
d000-d07f : PCI device 1011:0014
    d000-d07f : tulip
d400-d4ff : PCI device 1000:0001
    d400-d47f : ncr53c8xx
d800-d81f : PCI device 10b5:9050
    d800-d81f : TPMC551 (PCI)
e000-e07f : PCI device 10b5:9050
e800-e80f : PCI device 8086:7010
    e800-e807 : ide0
    e808-e80f : ide1
```

```
#cat /proc/iomem
00000000-0009ffff : System RAM
000a0000-000bffff : Video RAM area
000c0000-000c7fff : Video ROM
000f0000-000fffff : System ROM
00100000-03ffffff : System RAM
    00100000-002327d1 : Kernel code
    002327d2-0031bdcb : Kernel data
e4000000-e4ffffff : PCI device 1002:4758
e5800000-e580007f : PCI device 1011:0014
    e5800000-e580007f : tulip
e6000000-e60000ff : PCI device 1000:0001
e6800000-e680003f : PCI device 10b5:9050
e7000000-e700007f : PCI device 10b5:9050
ffff0000-ffffffff : reserved
```