# TPMC551-SW-82

## Linux Device Driver

8/4 Channel 16 Bit DAC

Version 1.2.x

## User Manual

Issue 1.2.3

February 2012

**TPMC551-SW-82**

Linux Device Driver

8/4 Channel 16 Bit DAC

| Issue | Description | Date |
|-------|-------------|------|
| 1.0 | First Issue | April 22, 2002 |
| 1.1 | Parameter for TP551_IOCGREADPARAM changed | May 13, 2002 |
| 1.2.0 | Kernel 2.6.x Support | March 21, 2005 |
| 1.2.1 | File list Changed, description of installation modified General revision | August 15, 2006 |
| 1.2.2 | New Address TEWS LLC | September 20, 2006 |
| 1.2.3 | General Revision | February 8, 2012 |

# Table of Contents

# 1 Introduction

The TPMC551-SW-82 Linux device driver allows the operation of the TPMC551 PMC conforming to the Linux I/O system specification. This includes a device-independent basic I/O interface with *open()*, *close(), write()* and *ioctl()* functions.

Special I/O operation that do not fit to the standard I/O calls will be performed by calling the *ioctl()* function with a specific function code and an optional function dependent argument.

The TPMC551-SW-82 device driver supports the following features:

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


The TPMC551-SW-82 device driver supports the modules listed below:

| | | |
|---|---|---|
| TPMC551 | 8/4 Channels of Isolated 16 bit D/A | (PMC) |


To get more information about the features and usage of TPMC551 devices it is recommended to read the manuals listed below.

| |
|---|
| TPMC551 User Manual |
| TPMC551 Engineering Manual |

# 2 Installation

Following files are located on the distribution media:

Directory path 'TPMC551-SW-82':

| | |
|---|---|
| TPMC551-SW-82-1.2.3.pdf | This manual in PDF format |
| TPMC551-SW-82-SRC.tar.gz | GZIP compressed archive with driver source code |
| ChangeLog.txt | Release history |
| Release.txt | Release information |

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

Directory path './tpmc551/':

| | |
|---|---|
| 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 |
| makenode | Script to create device nodes on the file system |
| include/config.h | Driver independent library header file |
| include/tpmodule.h | Driver and kernel independent library header file |
| include/tpmodule.c | Driver and kernel independent library source file |
| include/tpxxxhwdep.h | HAL library header file |
| include/tpxxxhwdep.c | HAL library source file |
| example/tpmc551exa.c | Example application |
| example/Makefile | Example application make file |

In order to perform an installation, extract all files of the archive TPMC551-SW-82.tar.gz to the desired target directory. The command 'tar -xzvf TPMC551-SW-82-SRC.tar.gz' will extract the files into the local directory.

- Login as *root* and change to the target directory

- Copy tpmc551.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 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 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 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 or sysfs (udev), 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 TPMC551 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 fd;


fd = open("/dev/tpmc501_0", O_RDWR);
if (fd < 0)
{
  /* handle open error conditions */
}
```

### RETURNS

The normal 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

| Error code | Description |
|------------|-------------|
| 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 fd;

if (close(fd) != 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

| Error code | Description |
|---|---|
| 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>
#include <tpmc551.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 are passed by the parameters *buffer* and *size* to the device.

```
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. <br><br> If this flag is not set, the output value will not be corrected. |
| TP551_FL_LATCHED | It this flag is set the data will be loaded into the DAC, but the conversion will not be started, until the *TP551_FL_SIMCONV* 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

```
#include "tpmc551.h"

int             hCurrent;
ssize_t         NumBytes;
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

| Error code | Description |
|------------|-------------|
| 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>
#include <tpmc551.h>

int ioctl
(
      int    filedes,
      int    request,
      void  *argp
)

## DESCRIPTION

The **ioctl** function sends a control code directly to a device, specified by *filedes*, 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.h*:

| Value | Meaning |
|---|---|
| TP551_IOCGREADPARAM | Get module parameters including the factory programmed correction values, number of channels, voltage range selection. |
| TP551_IOCSSTOPSEQ | Stop the sequencer |
| TP551_IOCSSTARTSEQ | Setup and start the sequencer |
| TP551_IOCSWRITESEQ | Write DAC data into sequencer FIFO-buffer |

See below for more detailed information on each control code.

**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

| Error code | Description |
|---|---|
| EINVAL | Invalid argument. This error code is also returned if the requested ioctl function is unknown. Please check the argument request. |

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

## 3.4.1　TP551_IOCGREADPARAM

### NAME

TP551_IOCGREADPARAM - Get the module parameters

### DESCRIPTION

This ioctl function returns the module 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.

```
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, which 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, which 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

```
#include <tpmc551.h>

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

…

result = ioctl(hCurrent, TP551_IOCGREADPARAM, &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.GainCorr[x]);
   }
}
else
{
  printf("\nRead module parameter failed -->  Error = %d\n",
    errno);
}

…
```

## ERRORS

| Error code | Description |
|---|---|
| EFAULT | Invalid pointer to the parameter buffer. Please check the argument argp. |

## 3.4.2    TP551_IOCSSTOPSEQ

### NAME

TP551_IOCSSTOPSEQ – Stop Sequencer Mode

### DESCRIPTION

This ioctl function stops the sequencer mode.

The optional argument can be omitted for this ioctl function.

### EXAMPLE

```
#include <tpmc551.h>

int  hCurrent;
int  result;

…

result = ioctl(hCurrent, TP551_IOCSSTOPSEQ);
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.

### 3.4.3 TP551_IOCSSTARTSEQ

#### NAME

TP551_IOCSSTARTSEQ - 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.

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 TP551_FL_CONTINUOUS is set the parameter will be ignored (see below).

*flags*

    This 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

```
#include <tpmc551.h>

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_IOCSSTARTSEQ, &SeqStartBuf);
if (result >= 0)
{
  printf("\nStarting sequencer successful\n");
}
else
{
  printf("\nStarting sequencer failed -->  Error = %d\n",
    errno);
}

…
```

## ERRORS

| Error code | Description |
|---|---|
| EFAULT | Invalid pointer to the parameter buffer. Please check the argument *argp*. |

### 3.4.4 TP551_IOCSWRITESEQ

#### NAME

TP551_IOCSWRITESEQ - Write DAC data into sequencer FIFO-buffer

#### DESCRIPTION

This ioctl function writes data into the sequencer FIFO. The 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.

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. Channels which are activated and not specified to be updated will hold their value.

*correction*

    This parameter specifies which channels shall use 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. 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

```c
#include <tpmc551.h>

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_IOCSWRITESEQ, &SeqWriteBuf);
if (result >= 0)
{
  printf("\nWriting sequencer data successful\n");
}
else
{
  printf("\nWriting sequencer data failed -->  Error = %d\n",
    errno);
}

…
```

## ERRORS

| Error code | Description |
|---|---|
| EFAULT | Invalid pointer to the parameter buffer. Please check the argument *argp*. |

# 4 <u>Diagnostic</u>

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 display information of a correctly running TPMC551-SW-82 device driver (see also the proc man pages).

```
# lspci -v
…
04:02.0 Signal processing controller: PLX Technology, Inc. PCI <-> IOBus
Bridge (rev 01)
     Subsystem: TEWS Technologies GmbH Device 0227
     Flags: medium devsel, IRQ 17
     Memory at feb9fc00 (32-bit, non-prefetchable) [size=128]
     I/O ports at e880 [size=128]
     I/O ports at e800 [size=32]
     Memory at feb9f800 (32-bit, non-prefetchable) [size=64]
     Kernel driver in use: TEWS TECHNOLOGIES - TPMC551 8 Channel 16 Bit DAC
     Kernel modules: tpmc551drv
…


# cat /proc/devices
Character devices:
…
254 tpmc551drv
…


#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
```