TPMC550-SW-72
LynxOS Device Driver
TPMC550 – 8 Channel 12-Bit DAC

Version 1.0.x

Reference Manual
Issue 1.0

April 2002

TEWS TECHNOLOGIES GmbH
Am Bahnhof 7
D-25469 Halstenbek
Germany
Tel.: +49 (0)4101 4058-0
Fax.: +49 (0)4101 4058-19
http://www.tews.com
e-mail: info@tews.com

TPMC550-SW-72
8 Channel 12-Bit DAC
LynxOS Device Driver

Issue	Description	Date
1.0	First Issue	April 5, 2002

Table of Contents

1 Introduction

The TPMC550-SW-72 LynxOS device driver allows the operation of a TPMC550 8 channel 12-bit DAC PMC on a PowerPC platform with DRM based PCI interface.

The standard file (I/O) functions (open, close, write and ioctl) provide the basic interface for opening and closing a file descriptor and for performing device I/O and control operations.

The TPMC550 device driver includes the following functions:

- write a new value to the specified DAC channel
- setup and start the DAC sequencer mode
- write data to FIFO-buffer for sequencer mode
- read module information
- all writes can be made with data correction using the factory set correction data

2 Installation

The software is delivered on a PC formatted 3½" HD diskette.

Following files are located on the diskette:

tpmc550.c	Driver source code
tpmc550.h	Definitions and data structures for driver and application
tpmc550_info.c	Device information definition
tpmc550_info.h	Device information definition header
tpmc550.cfg	Driver configuration file include
tpmc550.import	Linker import file
Makefile	Device driver make file
Makefile.dldd	Make file for dynamic driver installation
tpmc550-sw-72.pdf	This Manual in PDF format

2.1 Device Driver Installation

The two methods of driver installation are as follows:

- Static Installation
- Dynamic Installation (only native LynxOS systems)

2.1.1 Static Installation

With this method, the driver object code is linked with the kernel routines and is installed during system start-up.

In order to perform a static installation, copy the following files to the target directories:

1. Create a new directory in the system drivers directory path.
   For example: */sys/drivers.pp_drm/tpmc550*
2. Copy the following files to this directory: *tpmc550.c, Makefile*
3. Copy *tpmc550.h* to */usr/include/*
4. Copy *tpmc550_info.c* to */sys/devices/*
5. Copy *tpmc550_info.h* to */sys/dheaders/*
6. Copy *tpmc550.cfg* to */sys/cfg.ppc/*

2.1.1.1 Build the driver object

1. Change to the directory */sys/drivers.pp_drm/tpmc550*
2. To update the library */sys/lib/libdrivers.a* enter:

```
make install
```

2.1.1.2 Create Device Information Declaration

1. Change to the directory */sys/devices/*
2. Add the following dependencies to the *Makefile*

   ```
   DEVICE_FILES_prep = ...tpmc550_info.x
   ```

   And at the end of the Makefile

   ```
   ...
   tpmc550_info.o:$(DHEADERS)/tpmc550_info.h
   ```

3. To update the library */sys/lib/libdevices.a* enter:

   ```
   make install
   ```

2.1.1.3 Modify the Device and Driver Configuration File

In order to insert the driver object code into the kernel image, an appropriate entry in file CONFIG.TBL must be created.

1. Change to the directory /sys/lynx.os/
2. Create an entry in the file CONFIG.TBL
   Insert the entry after the console driver section

   ```
   # End of console devices
   I:tpmc550.cfg
   ```

2.1.1.4 Rebuild the Kernel

1. Change to the directory /sys/lynx.os/ (/sys/bsp.pp_drm)
2. To rebuild the kernel enter the following command:

   ```
   make install
   ```

3. Reboot the newly-created operating system by the following command:

   ```
   reboot –aN
   ```

   The **N** flag instructs *init* to run *mknod* and create all the nodes mentioned in the new *nodetab*.

4. After reboot you should find the following new devices (depends on the device configuration): */dev/tp550a, [/dev/tp550b, …]*

2.1.2 Dynamic Installation

This method allows you to install the driver after the operating system is booted. The driver object code is attached to the end of the kernel image and the operating system dynamically adds this driver to its internal structures. The driver can also be removed dynamically.
Unlike the description of the dynamic installation in the manual “Writing Device Drivers for LynxOS”, the driver source must be placed in a directory under */sys/drivers.pp_drm/*

The following steps describe how to do a dynamic installation:

1. Create a new directory in the system drivers directory path.
   For example: */sys/drivers.pp_drm/tpmc550*

2. Copy the following files to this directory:
   - tpmc550.c
   - tpmc550_info.c
   - tpmc550_info.h
   - tpmc550.import
   - Makefile.dldd

3. Copy *tpmc550.h* to */usr/include*

4. Change to the directory */sys/drivers.pp_drm/tpmc550*

5. To make the dynamic link-able driver enter :
   ```
   make –f Makefile.dldd
   ```

6. Create a device definition file for one major device
   ```
   gcc –DDLDD –o tpmc550_info tpmc550_info.c
   ./tpmc550_info > tp550a
   ```

7. To install the driver enter:
   ```
   drinstall –c tpmc550.obj
   ```

   If successful *drinstall* returns a unique <*driver-ID*>

8. To install the major device enter:
   ```
   devinstall –c –d <driver-ID> tp550a
   ```

   The <*driver-ID*> is returned by the *drinstall* command

9. To create nodes for the devices enter:
   ```
   mknod /dev/tp550a c <major_no> 0
   ...
   ```

If all steps are successful completed the TPMC550 is ready to use.

To uninstall the TPMC550 device enter the following commands:
```
devinstall –u –c <device-ID>
drinstall –u <driver-ID>
```

2.1.3 Device Information Definition File

The device information definition contains information necessary to install the TPMC550 major device.
The implementation of the device information definition is done through a C structure which is defined in the header file *tpmc550_info.h*.

This structure contains following parameter:

PCIBusNumber Contains the PCI bus number at which the TPMC550 is connected. Valid bus numbers are in range from 0 to 255.

PCIDeviceNumber Contains the device number (slot) at which the TPMC550 is connected. Valid device numbers are in range from 0 to 31.

NOTE. If both *PCIBusNumber* and *PCIDeviceNumber* are –1 then the driver will auto scan for the TPMC550 device. The first device found in the scan order will be allocated by the driver for this major device.
Already allocated devices can't be allocated twice. This is important to know if you have more than one TMPC550 major device.

A device information definition is unique for every TPMC550 major device. The file *tpmc550_info.c* on the distribution disk contains two device information declarations, **tp550a_info** for the first major device and **tp550b_info** for the second major device.

If the driver should support more than two major devices it is necessary to copy and paste an existing declaration and rename it with unique name for example **tp550c_info**, **tp550d_info** and so on.

NOTE. It is also necessary to modify the device and driver configuration file respectively the configuration include file *tpmc550.cfg*.

The following device declaration information uses the auto find method to detect the TPMC550 module on PCI bus.

```
TP550_INFO tp550a_info = {

    -1,                 /*  auto find the TPMC550 on any PCI bus    */
    -1,

};
```

2.1.4 Configuration File: CONFIG.TBL

The device and driver configuration file *CONFIG.TBL* contains entries for device drivers and its major and minor device declarations. Each time the system is rebuild, the *config* utility reads this file and produces a new set of driver and device configuration tables and a corresponding *nodetab*.

To install the TPMC550 driver and devices into the LynxOS system, the configuration include file *tpmc550.cfg* must be included in the *CONFIG.TBL* (see also 2.1.1.3).
The file *tpmc550.cfg* on the distribution disk contains the driver entry (C:tpmc550:\....) and one enabled major device entry (D:TPMC550 1:tp550a_info::) with one minor device entry (N: tp550a:0).

If the driver should support more than one major device the following entries for major and minor devices must be enabled by removing the comment character (#). By copy and paste an existing major and minor entry and renaming the new entries, it is possible to add any number of additional TPMC550 device.

NOTE. The name of the device information declaration (info-block-name) must match to an existing C structure in the file *tpmc550_info.c*.

This example shows a driver entry with one major device and 8 minor devices:

```
#Format:
#C:driver-name:open:close:read:write:select:control:install:uninstall
#D:device-name:info-block-name:raw-partner-name
#N:node-name:minor-dev

C:tpmc550:\
  :tp550open:tp550close:tp550read::\
  ::tp550ioctl:tp550install:tp550uninstall
D:TPMC550 1:tp550a_info::
N:tp550a:0
```

The configuration above creates the following node in the */dev* directory.

```
/dev/tp550a
```

3 TPMC550 Device Driver Programming

LynxOS system calls are all available directly to any C program. They are implemented as ordinary function calls to "glue" routines in the system library, which trap to the OS code.
Note that many system calls use data structures, which should be obtained in a program from appropriate header files. Necessary header files are listed with the system call synopsis.

3.1 open()

NAME

open() - open a file

SYNOPSIS

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

int open (char *path, int oflags[, mode_t mode])

DESCRIPTION

Opens a file (TPMC550 device) named in ***path*** for reading and writing. The value of ***oflags*** indicates the intended use of the file. In case of a TPMC550 devices ***oflags*** must be set to **O_WRONLY** to open the file for writing.
The ***mode*** argument is required only when a file is created. Because a TPMC550 device already exists this argument is ignored.

EXAMPLE

```
int  fd

…

/*
**  open the device named "/dev/tp550a" for Input
*/

fd = open ("/dev/tp550a", O_WRONLY);

…
```

RETURNS

open returns a file descriptor number if successful, or –1 on error.

SEE ALSO

LynxOS System Call - open()

3.2 close()

NAME

close() – close a file

SYNOPSIS

int close(int fd)

DESCRIPTION

This function closes an opened device.

EXAMPLE

```
int               result;

…

/*
**     close the device
*/

result = close(fd);

…
```

RETURNS

close returns 0 (OK) if successful, or –1 on error

SEE ALSO

LynxOS System Call - close()

3.3 write()

NAME

write() - write to a file

SYNOPSIS

#include <tpmc550.h>

int write (int fd, char *buff, int count)

DESCRIPTION

The write function writes a DAC value to the specified channel.
The argument ***buff*** contains a pointer to the write buffer (TP550_WRITE_BUFFER), which contains information for the desired write operation.
The argument ***count*** is not required and should be 0.

The *TP550_WRITE_BUFFER* structure has the following layout:

```
typedef struct
{
    unsigned short      channel;        /* channel number */
    unsigned short      flags;
    short               value;          /* DAC output value */
} TP550_WRITE_BUFFER, *PTP550_WRITE_BUFFER;
```

The parameter **channel** specifies the DAC channel that will be used. Allowed values are 1 to 8 for TPMC550-10/-20 and 1 to 4 for TPMC550-11/-21.

The parameter **flags** value is an ORed value of the flags shown in the following table.

Name	Meaning
TP550_FL_CORR	If this flag is set, the driver will correct the DAC output value with the factory programmed correction data. Otherwise the data will be written directly.
TP550_FL_LATCHED	The DAC output values will be latched. The output must be activated with the *TP550_FL_SIMCONV* flag. If the *TP550_FL_SIMCONV* is not set, the data will be just written to the channel, but the conversion will not be started.
TP550_FL_SIMCONV	This flag must be set to start a parallel conversion on all channels. This flag allows to output latched values.

The parameter **value** specifies the DAC output value. The values must be between 0 and 4095 for 0V..+10V mode and between –2048 and +2047 for –10V..+10V mode.

EXAMPLE

```
int                         fd;
int                         result;
unsigned short              value;
TP550_WRITE_BUFFER          WriteBuf;

…

/*****************************************
 Write new values to channel 5 and channel 2
 correct the output data for channel 5
 latch values and make a parallel output
 conversion.
*****************************************/
WriteBuf.channel      = 5;
WriteBuf.flags        = TP550_FL_CORR | TP550_FL_LATCHED;
WriteBuf.value        = 1024;

result = write(fd, (char*)& WriteBuf, 0);
if( result == sizeof(TP550_WRITE_BUFFER)) {
  printf("Write OK\n");
}
else {
  printf( "\nWrite failed --> Error = %d.\n", errno );
}

WriteBuf.channel      = 2;
WriteBuf.flags        = TP550_FL_LATCHED | TP550_FL_SIMCONV;
WriteBuf.value        = 512;

result = write(fd, (char*)& WriteBuf, 0);
if( result == sizeof(TP550_WRITE_BUFFER)) {
  printf("Write OK\n");
}
else {
  printf( "\nWrite failed --> Error = %d.\n", errno );
}

…
```

RETURNS

When *write* succeeds, the size of the write buffer is returned. If write fails, -1 (SYSERR) is returned.

On error, *errno* will contain a standard read error code (see also LynxOS System Call – read) or one of the following TPMC550 specific error codes:

ENXIO	Invalid minor device specified.
EBUSY	Sequencer mode is active. This function can not be called while the sequencer mode is active.
ETIMEDOUT	The maximum allowed time to finish the read request is exhausted.
EINVAL	Invalid parameter. Please check the parameter.

SEE ALSO

LynxOS System Call - read()

3.4 ioctl()

NAME

ioctl() - I/O device control

SYNOPSIS

#include <ioctl.h>
#include <tpmc550.h>

int ioctl (int fd, int request, char *arg)

DESCRIPTION

ioctl provides a way of sending special commands to a device driver. The call sends the value of ***request*** and the pointer ***arg*** to the device associated with the descriptor ***fd***.

The following request values are support by a TPMC550 device :

Value	**Meaning**
TP550_READPARAM	Read module parameters, this includes the model type, the correction data, the number of channels and the output range selection.
TP550_SEQSTOP	Stop sequencer, set module to normal mode.
TP550_SEQSTART	Setup and start sequencer, set module in sequencer mode.
TP550_SEQWRITE	Write data to sequencer buffer.

See behind for more detailed information on each control code.

Note

To use these TPMC550 specific control codes the header file tpmc550.h must be included in the application.

RETURNS

ioctl returns 0 if successful, or –1 on error.
The TPMC550 ioctl function returns always standard error codes. See LynxOS system call ioctl of a detailed description of possible error codes.

SEE ALSO

LynxOS System Call - ioctl().

3.4.1 TP550_READPARAM - (Read Module Parameters)

The function *TP550_READPARAM* reads the module parameters of the TPMC550 including the model, the correction data, the number of channels and the output range selection.
The argument ***arg*** contains a pointer to the *TP550_PARA_BUFFER* data structure.

The *TP550_PARA_BUFFER* structure has the following layout:

```
typedef struct
{
    int             ModuleType;     /* TPMC550 variant type */
    int             NumChans;       /* Number of channels */
    int             vMode1_4;       /* Voltage Mode 1..4 */
    int             vMode5_8;       /* Voltage Mode 5..8, if present */
    signed char     OffsCorr[8];    /* Offset correction Data */
    signed char     GainCorr[8];    /* Gain correction Data */
} TP550_PARA_BUFFER, *PTP550_PARA_BUFFER;
```

The entry **ModuleType** returns the model type. A value of 10 specifies a TPMC550-10, a value of 11 specifies a TPMC550-11 and so on.

The entry **NumChans** returns the number of channels supported by the module.

The entry **vMode1_4** returns TRUE or FALSE. If TRUE the channels 1 to 4 are configured for –10V..+10V output range. If FALSE the channels 1 to 4 are configured for 0V..+10V output range.

The entry **vMode5_8** returns TRUE or FALSE. If TRUE the channels 5 to 8 are configured for –10V..+10V output range. If FALSE the channels 5 to 8 are configured for 0V..+10V output range.

The array **OffsCorr** returns the offset correction data. The index of the array specifies the channel the value is assigned to. Index 0 for channel 1, index 1 for channel 2 and so on.

The array **GainCorr** returns the gain correction data. The index of the array specifies the gain the value is assigned to. Index 0 for channel 1, index 1 for channel 2 and so on.

Note

More information about data correction is printed in the
TPMC550 User Manual

EXAMPLE

```
int                   fd;
int                   result;
TP550_PARA_BUFFER     ParamBuf;

…

/*
**  Read module parameters
*/
result = ioctl (fd, TP550_READPARAM, (char*)&ParamBuf);

if (result < 0) {
  /* handle ioctl error */
}
else {
  printf("\nModule type = TPMC550-%02d\n",
           ParamBuf.ModuleType);
  printf("\nChannels    = %d\n",
           ParamBuf.NumChans);
  printf("\nRange 1..4  = %3dV..+10V \n",
           ParamBuf.vMode1_4 ? -10 : 0);
  printf("\nRange 5..8  = %3dV..+10V \n",
           ParamBuf.vMode5_8 ? -10 : 0);
  printf("Offset Error = %d, %d, %d, %d, %d, %d, %d, %d\n",
           ParamBuf.OffsCorr[0],
           ParamBuf.OffsCorr[1],
           ParamBuf.OffsCorr[2],
           ParamBuf.OffsCorr[3],
           ParamBuf.OffsCorr[4],
           ParamBuf.OffsCorr[5],
           ParamBuf.OffsCorr[6],
           ParamBuf.OffsCorr[7]);
  printf("Gain Error   = %d, %d, %d, %d, %d, %d, %d, %d\n",
           ParamBuf.GainCorr[0],
           ParamBuf.GainCorr[1],
           ParamBuf.GainCorr[2],
           ParamBuf.GainCorr[3],
           ParamBuf.GainCorr[4],
           ParamBuf.GainCorr[5],
           ParamBuf.GainCorr[6],
           ParamBuf.GainCorr[7]);
}

…
```

3.4.2 TP550_SEQSTOP - (Stop Sequencer Mode)

The function *TP550_SEQSTOP* stops the sequencer and returns the module to normal mode. The last value written to the channels will be held at the output.
The argument ***arg*** is unused and should be set to zero.

EXAMPLE

```
int                   fd;
int                   result;

…

/*
**  Get module parameters
*/
result = ioctl (fd, TP550_SEQSTOP, 0);
if (result < 0) {
  /* handle ioctl error */
}

…
```

3.4.3 TP550_SEQSTART - (Setup and Start Sequencer Mode)

The function *TP550_SEQSTART* sets up the TPMC550 to work in sequencer mode. The cycle time and the channel configuration are set up.
The argument ***arg*** contains a pointer to the TP550_ST_SEQ_BUFFER data structure.

The TP550_ST_SEQ_BUFFER structure has the following layout:

```
typedef struct
{
    unsigned short      channels;       /* channel selection */
    unsigned short      cycleTime;      /* cycle time */
    unsigned short      flags;          /* flags */
} TP550_ST_SEQ_BUFFER, *PTP550_ST_SEQ_BUFFER;
```

The entry **cycleTime** 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. This value will not be used if *TP550_FL_HANDSHAKE* is specified.

The argument **channel** specifies the channels that shall be used in sequencer mode. If bit 0 is set channel 1 will be used, if channel 2 shall be used, bit 1 must be set, and so on.

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

Name	Meaning
TP550_FL_LATCHED	If this flag is set, the driver will use output the data in latched mode, the data will be visible at the same time. Otherwise the data will be used in transparent mode.
TP550_FL_HANDSHAKE	The sequencer will work in handshake mode. The data will be written when new data is written with the *TP550_SEQWRITE* command.

EXAMPLE

```
int                   fd;
int                   result;
TP550_ST_SEQ_BUFFER   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 = TP550_FL_LATCHED;

result = ioctl (fd, TP550_SEQSTART, (char*)&SeqStartBuf);

if (result < 0) {
  /* handle ioctl error */
}

…
```

3.4.4 TP550_SEQWRITE - (Write Sequencer Data)

The function *TP550_SEQWRITE* writes DAC data for sequencer output. This function writes data to the sequencer software-FIFO if the timer mode is selected, if the handshake mode is enabled, the data will be written directly to the sequencer data registers and if the sequencer is stopped, the data will be stored as default start values. If timer mode is active, the data is stored in a FIFO and will be written during a function which is called when the sequencer indicates, that new data can be written. The clock rate is defined with the TP550_SEQSTART command.
If handshake mode is enabled, this function will trigger the sequencer. The refresh time for new output values depends on the application.
The argument ***arg*** contains a pointer to the *TP550_WR_SEQ_BUFFER* data structure.

The *TP550_WR_SEQ_BUFFER* 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 */
} TP550_WR_SEQ_BUFFER, *PTP550_WR_SEQ_BUFFER;
```

The argument **channels** specifies the channels that shall be updated with this data set. If bit 0 is set channel 1 will be updated, if channel 2 shall be updated, bit 1 must be set, and so on. Channel which are active and not specified to be updated, will held their value.

The argument **correction** specifies the channels that shall use corrected output values. If bit 0 is set channel 1 will be use corrected data, if channel 2 shall use corrected data, bit 1 must be set, and so on. Data correction uses the factory stored correction data.

The array **values** 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 4095 for 0V..+10V mode and between –2048 and +2047 for –10V..+10V mode. Only the values for channels specified for update will be used.

The argument **stat** returns the sequencer status. The status returns number of cycles which had not been used for new data output, because the 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                   fd;
int                   result;
TP550_WR_SEQ_BUFFER   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] = 0x123;
SeqWriteBuf.values[5] = 0x700;

result = ioctl (fd, TP550_SEQWRITE, (char*)&SeqWriteBuf);

if (result < 0) {
  /* handle ioctl error */
}
else {
  /* Check SeqWriteBuf.stat */
}

…
```

4 Debugging

This driver was successful tested on a Motorola MVME3600-1 (PMCSPAN) and MVME2305-900 board in a native LynxOS environment and a Windows Cross development.

If the driver will not work properly, usually a PCI bus or interrupt problem, you can enable debug outputs by removing the comments around the symbols *DEBUG*, *DEBUG_PCI* and *DEBUG_TPMC*. The debug output will appear on the console.

The debug output displays the PCI Header, the address of each base address register and a memory dump of all mapped memory and I/O spaces of the TPMC550 like this (see also *TPMC550 User Manual – PCI Configuration*).

```
TPMC550 Device Driver Install
Bus = 0  Dev = 16  Func = 0
[00] = 905010B5
[04] = 02800000
[08] = 11800001
[0C] = 00000008
[10] = 02042000
[14] = 0000C001
[18] = 0000D001
[1C] = 02043000
[20] = 00000000
[24] = 00000000
[28] = 00000000
[2C] = 02261498
[30] = 00000000
[34] = 00000000
[38] = 00000000
[3C] = 00000109
PCI Base Address 0 (PCI_RESID_BAR0)

E8142000 : E1 FF FF 0F E0 FF FF 0F 00 00 00 00 00 00 00 00
E8142010 : 00 00 00 00 01 00 00 00 01 01 00 00 00 00 00 00
PCI Base Address 1 (PCI_RESID_BAR1)

E0108000 : E1 FF FF 0F E0 FF FF 0F 00 00 00 00 00 00 00 00
E0108010 : 00 00 00 00 01 00 00 00 01 01 00 00 00 00 00 00
PCI Base Address 2 (PCI_RESID_BAR2)

E0109000 : 00 00 00 00 00 0A 00 00 00 00 00 00 00 00 00 00
E0109010 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
PCI Base Address 3 (PCI_RESID_BAR3)

E8143000 : FF FE FA FA F7 FD FC FE 06 04 03 0E 09 0C 06 08
E8143010 : FF FE FD FC FB FE FD FE 03 02 02 07 05 06 03 04

Moduletype TPMC550-10
```