The Embedded I/O Company

TEWS TECHNOLOGIES

TPMC600-SW-82

Linux Device Driver

32/16 Digital Inputs (24V)

Version 1.0.x

User Manual

Issue 1.0.0

September 2009

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

TPMC600-SW-82

Linux Device Driver

32/16 Digital Inputs (24V)

Supported Modules:
TPMC600

Issue	Description	Date
1.0.0	First Issue	September 16, 2009

Table of Contents

1 INTRODUCTION 4

2 INSTALLATION 5

2.1 Build and install the device driver 5

2.2 Uninstall the device driver 6

2.3 Install device driver into the running kernel 6

2.4 Remove device driver from the running kernel 6

2.5 Change Major Device Number 6

2.6 Maximum Number of Active Jobs Configuration 7

3 I/O FUNCTIONS 8

3.1 open() 8

3.2 close() 10

3.3 ioctl() 11

3.3.1 TPMC600_IOCXREAD 13

3.3.2 TPMC600_IOCDEBENABLE 16

3.3.3 TPMC600_IOCDEBDISABLE 17

4 DIAGNOSTIC 18

1 Introduction

The TPMC600-SW-82 Linux device driver allows the operation of the TPMC600 Digital Input PMC conforming to the Linux I/O system specification. This includes a device-independent basic I/O interface with *open()*, *close()* and *ioctl()* functions.

The TPMC600-SW-82 device driver supports the following features:

- Reading digital input value immediately or after a selected event occurs
- Input hardware debouncing

The TPMC600-SW-82 device driver supports the modules listed below:

TPMC600-10	32 digital inputs (Front Panel I/O)	(PMC)
TPMC600-11	16 digital inputs (Front Panel I/O)	(PMC)
TPMC600-20	32 digital inputs (P14 I/O)	(PMC)
TPMC600-21	16 digital inputs (P14 I/O)	(PMC)

To get more information about the features and use of TPMC600 devices it is recommended to read the manuals listed below.

TPMC600 User Manual

TPMC600 Engineering Manual

2 Installation

Following files are located on the distribution media:

Directory path 'TPMC600-SW-82':

TPMC600-SW-82-SRC.tar.gz	GZIP compressed archive with driver source code
TPMC600-SW-82-1.0.0.pdf	PDF copy of this manual
ChangeLog.txt	Release history
Release.txt	Release information

The GZIP compressed archive TPMC600-SW-82-SRC.tar.gz contains the following files and directories:

Directory path 'tpmc600':

tpmc600.c	TPMC600 device driver source
tpmc600def.h	TPMC600 driver include file
tpmc600.h	TPMC600 include file for driver and application
Makefile	Device Driver Makefile
makenode	Script for Device Node Creation in File System
include/tpxxxhwdep.c	Hardware dependent library
include/tpxxxhwdep.h	Hardware dependent library header file
include/tpmodule.c	Driver independent library
include/tpmodule.h	Driver independent library header file
include/config.h	Driver independent library header file
example/tpmc600exa.c	Example Application
example/Makefile	Makefile for Example Application

In order to perform an installation, extract all files of the archive TPMC600-SW-82-SRC.tar.gz to the desired target directory. The command 'tar -xzvf TPMC600-SW-82-SRC.tar.gz' will extract the files into the local directory.

- Login as *root* and change to the target directory
- Copy tpmc600.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 tpmc600drv

- After the first build or if you are using dynamic major device allocation it is 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 TPMC600 module found. The first TPMC600 module can be accessed with device node /dev/tpmc600_0, the second module with device node /dev/tpmc600_1, and so on.

The assignment of device nodes to physical TPMC600 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 tpmc600drv

If your kernel has enabled a dynamic file system, all /dev/tpmc600_x nodes will be automatically removed from your file system after this.

Make sure that the driver isn't opened by any application program. If opened you will get the response "*tpmc600drv: 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 TPMC600 driver use dynamic allocation of major device numbers per default. If this isn't suitable for the application it is possible to define a major number for the driver.

To change the major number edit the file tpmc600def.h, change the following symbol to appropriate value and enter *make install* to create a new driver.

TPMC600_MAJOR	Valid numbers are in range between 0 and 255. A value of 0 means dynamic number allocation.

Example:

```
#define TPMC600_MAJOR        122
```

Be sure that the desired major number is not used by other drivers. Please check */proc/devices* to see which numbers are free.

Keep in mind that it is necessary to create new device nodes if the major number for the TPMC600 driver has changed and the makenode script is not used.

2.6 Maximum Number of Active Jobs Configuration

The maximum number of active event read jobs per major device can be configured. This can be simply made by changing the value of the symbol in tpmc600def.h.

NUM_REQUESTS	Defines the maximum number of active event wait jobs (default = 100). Valid numbers are in range between 1 and MAXINT.

3 I/O 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 is to be opened. This is a bit mask; you 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

```
#include <fcntl.h>

int  fd;

…

fd = open(“/dev/tpmc600_0”, O_RDWR);
if (fd == -1)
{
     /* handle error condition */
}
```

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

ENODEV	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

```
#include <unistd.h>

int  fd;

…

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

RETURNS

The normal 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.

3.3 ioctl()

NAME

ioctl() device control functions

SYNOPSIS

```
#include    <sys/ioctl.h>
#include    <tpmc600.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 *tpmc600.h*:

Value	Meaning
TPMC600_IOCXREAD	Read state of the digital input lines
TPMC600_IOCDEBENABLE	Configure and enable input debouncer
TPMC600_IOCDEBDISABLE	Disable input debouncer

See below for more detailed information on each control code.

To use these TPMC600 specific control codes the header file *tpmc600.h* must be included in the application.

RETURNS

On success, zero is returned. In 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 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 TPMC600 driver always returns standard Linux error codes.

SEE ALSO

ioctl man pages

3.3.1 TPMC600_IOCXREAD

NAME

TPMC600_IOCXREAD - Read state of input lines

DESCRIPTION

This function reads the state of the input lines. The read is performed immediately when calling the function or after a specified event has occurred. A pointer to the callers read buffer (*TPMC600_READBUF*) must be passed by the parameter *arg* to the device.

```
typedef struct
{
        unsigned int    value;      /* read state of input lines */
        unsigned int    mode;       /* read mode */
        unsigned int    mask;       /* mask for relevant bits */
        unsigned long   timeout;    /* timeout in ticks, 0 = wait indefinitely */
} TPMC600_READBUF;
```

Members

mode

Specifies when the input state will be read. The following read modes are supported, and appropriate defines can be found in tpmc600.h:

Mode	Description
TPMC600_NOW	The state of the input lines will be read immediately.
TPMC600_HIGH_TR	The function will wait for a low to high transition on any of the input lines specified in *mask*
TPMC600_LOW_TR	The function will wait for a high to low transition on any of the input lines specified in *mask*
TPMC600_ANY_TR	The function will wait for any transition on any of the input lines specified in *mask*

mask

This function specifies the input lines the driver will use to wait for the specified transition. A set bit specifies the appropriate input line. Bit-0 specifies IN 1, bit-1 specifies IN 2, and so on.
If the *mode* is set to *TPMC600_NOW* this parameter is unused.

timeout

This parameter specifies the maximum wait time (ticks), before the function terminates and returns with an error.
If the *mode* is set to *TPMC600_NOW* this parameter is unused.

value

This value returns the state of the input lines at the moment specified by *mode*. Bit-0 returns the state of IN 1, bit-1 the state of IN 2, and so on.

There is a delay between the occurrence of the specified event and reading the input value, which is based on the system and OS dependent interrupt latency.

EXAMPLE

```
#include <sys/ioctl.h>
#include <tpmc600.h>

int                fd;
int                result;
TPMC600_READBUF    ReadBuf;

…

/* Read input port immediately without waiting for any event */
ReadBuf.mode = TPMC600_NOW;
result = ioctl(fd, TPMC600_IOCXREAD, & ReadBuf);
if (result >= 0)
{
     printf(“Input port = 0x%x\n”, ReadBuf.value);
}
else
{
     /* handle read error */
}

…
```

…

```
/* Read the input port after a high-transition at bit 7 occurred */
ReadBuf.mode       = TPMC600_HIGH_TR;
ReadBuf.mask       = 1 << 7;          /* high-transition at bit 7 */
ReadBuf.timeout    = 100;             /* ticks */
result = ioctl(fd, TPMC600_IOCXREAD, & ReadBuf);
if (result >= 0)
{
     printf("Input port = 0x%x\n", ReadBuf.value);
}
else
{
     /* handle read error */
}
```

ERRORS

This ioctl function returns no function specific error codes.

SEE ALSO

ioctl man pages

3.3.2 TPMC600_IOCDEBENABLE

NAME

TPMC600_IOCDEBENABLE - enables input debouncer function

DESCRIPTION

This ioctl function enables the input debouncer function. The argument *argp* passes the new debouncer counter value to the driver.

Please refer to the corresponding TPMC600 device hardware user manual for calculation formulas for appropriate counter values.

EXAMPLE

```
#include <sys/ioctl.h>
#include <tpmc600.h>

int      fd;
int      result;

…

/* Enable the debouncer with a debounce time of ~1ms (147*7us)*/
result = ioctl(fd, TPMC600_IOCDEBENABLE , 147);
if (result < 0)
{
     /* handle ioctl error */
}
```

ERRORS

This ioctl function returns no function specific error codes.

SEE ALSO

ioctl man pages

3.3.3 TPMC600_IOCDEBDISABLE

NAME

TPMC600_IOCDEBDISABLE - disables input debouncer function

DESCRIPTION

This ioctl function disables the input debouncer function.

The optional argument can be omitted for this ioctl function.

EXAMPLE

```
#include <sys/ioctl.h>
#include <tpmc600.h>

int       fd;
int       result;

…

result = ioctl(fd, TPMC600_IOCDEBDISABLE);
if (result < 0)
{
     /* handle ioctl error */
}
```

ERRORS

This ioctl function returns no function specific error codes.

SEE ALSO

ioctl man pages

4 Diagnostic

If the TPMC600 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, drivers, devices and so on. The following screen dumps displays information of a correct running TPMC600 device driver (see also the proc man pages).

```
# cat /proc/pci
PCI devices found:
…
  Bus  0, device  16, function  0:
    Signal processing controller: PCI device 1498:0258 (TEWS Datentechnik
GmBH) (rev 0).
      IRQ 25.
      Non-prefetchable 32 bit memory at 0xbffbff80 [0xbffbffff].
      I/O at 0xbfef00 [0xbfef7f].
      Non-prefetchable 32 bit memory at 0xbffbff60 [0xbffbff7f].
```

```
# cat /proc/devices
Character devices:
  1 mem
  2 pty/m%d
  3 pty/s%d
  4 tts/%d
  5 cua/%d
 10 misc
128 ptm
136 pts/%d
162 raw
254 tpmc600drv
```

```
# cat /proc/ioports
00000000-00bfffff : PCI host bridge
  00bfef00-00bfef7f : PCI device 1498:0258 (TEWS Datentechnik GmBH)
  00bfefc0-00bfefff : Intel Corp. 82559ER
    00bfefc0-00bfefff : eepro100
  00bff000-00bfffff : Tundra Semiconductor Corp. CA91C042 [Universe]
ffe80000-ffe80007 : serial(auto)
ffe80008-ffe8000f : serial(auto)
```

```
# cat /proc/iomem
80000000-ffffffff : PCI host bridge
  80000000-9fffffff : Universe VMEbus
    80000000-8000ffff : VME Master 0
    80010000-8020ffff : VME Master 1
  bffbff60-bffbff7f : PCI device 1498:0258 (TEWS Datentechnik GmBH)
  bffbff80-bffbffff : PCI device 1498:0258 (TEWS Datentechnik GmBH)
  bffc0000-bffdffff : Intel Corp. 82559ER
  bfffe000-bfffefff : Intel Corp. 82559ER
    bfffe000-bfffefff : eepro100
  bffff000-bfffffff : Tundra Semiconductor Corp. CA91C042 [Universe]
```