The Embedded I/O Company

TEWS TECHNOLOGIES

TPMC600-SW-72

LynxOS Device Driver

32/16 Digital Inputs (24V)

Version 1.0.x

User Manual

Issue 1.0.0

August 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-72

LynxOS Device Driver

32/16 Digital Inputs (24V)

Supported Modules:
TPMC600

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.

©2009 by TEWS TECHNOLOGIES GmbH

Issue	Description	Date
1.0.0	First Issue	August 18, 2009

Table of Contents

1 Introduction

The TPMC600-SW-72 LynxOS device driver allows the operation of the TPMC600 Digital Input PMC conforming to the LynxOS I/O system specification. This includes a device-independent basic I/O interface with *open()*, *close()* and *ioctl()* functions.

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

The TPMC600-SW-72 device driver supports the following features:

- reading current state of input lines (immediately)
- waiting for events on the input lines and than reading the state of the input lines
- configuring input debouncer

The TPMC600-SW-72 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-72’:

TPMC600-SW-72-SRC.tar.gz	GZIP compressed archive with driver source code
TPMC600-SW-72-1.0.0.pdf	PDF copy of this manual
ChangeLog.txt	Release history
Release.txt	Release information

For installation the files have to be copied to the desired target directory.

The GZIP compressed archive TPMC600-SW-72-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
tpmc600_info.c	TPMC600 Device information definition
tpmc600_info.h	TPMC600 Device information definition header
tpmc600.cfg	TPMC600 Driver configuration file include
tpmc600.import	Linker import file
Makefile	Device driver make file
example/tpmc600exa.c	Example application
example/Makefile	Example application makefile

In order to perform a driver installation, first extract the TAR file to a temporary directory, than follow the steps below:

1. Create a new directory in the system drivers directory path /sys/drivers.xxx, where xxx represents the BSP that supports the target hardware.

   For example: /sys/drivers.pp_drm/tpmc600 or /sys/drivers.cpci_x86/tpmc600

2. Copy the following files to this directory:
   - tpmc600.c
   - tpmc600def.h
   - tpmc600.import
   - Makefile

3. Copy tpmc600.h to /usr/include/

4. Copy tpmc600_info.c to /sys/devices.xxx/ or /sys/devices if /sys/devices.xxx does not exist (xxx represents the BSP).

5. Copy tpmc600_info.h to /sys/dheaders/

Copy tpmc600.cfg to */sys/cfg.xxx/*, where xxx represents the BSP for the target platform. For example: /sys/cfg.ppc or /sys/cfg.x86 ....

2.1 Device Driver Installation

The two methods of driver installation are as follows:

(1) Static Installation
(2) Dynamic Installation (only native LynxOS 4 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.

2.1.1.1 Build the driver object

1. Change to the directory /sys/drivers.xxx/tpmc600, where xxx represents the BSP that supports the target hardware.
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.xxx/ or /sys/devices if /sys/devices.xxx does not exist (xxx represents the BSP).
2. Add the following dependencies to the Makefile

   ```
   DEVICE_FILES_all = … tpmc600_info.x
   ```

   And at the end of the Makefile

   ```
   tpmc600_info.o:$(DHEADERS)/tpmc600_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/ respective /sys/bsp.xxx, where xxx represents the BSP that supports the target hardware.
2. Create an entry at the end of the file CONFIG.TBL

   Insert the following entry at the end of this file.

   ```
   I:tpmc600.cfg
   ```

2.1.1.4 Rebuild the Kernel

1. Change to the directory /sys/lynx.os/ (/sys/bsp.xxx)
2. Enter the following command to rebuild the kernel:

   ```
   make install
   ```

3. Reboot the newly created operating system by the following command (not necessary for KDIs):

   ```
   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/tpmc600a, /dev/tpmc600b, …

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.

2.1.2.1 Build the driver object

1. Change to the directory /sys/drivers.xxx/tpmc600, where xxx represents the BSP that supports the target hardware.
2. To make the dynamic link-able driver enter:

   ```
   make dldd
   ```

2.1.2.2 Create Device Information Declaration

1. Change to the directory /sys/drivers.xxx/tpmc600, where xxx represents the BSP that supports the target hardware.
2. To create a device definition file for the major device (this works only on native systems)

   ```
   make t600info
   ```

3. To install the driver enter:

   ```
   drinstall –c tpmc600.obj
   ```

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

4. To install the major device enter:

   ```
   devinstall –c –d <driver-ID> t600info
   ```

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

5. To create the node for the devices enter:

   ```
   mknod /dev/tpmc600a c <major_no> 0
   ```

   The <major_no> is returned by the devinstall command.

If all steps are successfully completed, the TPMC600 is ready to use.

2.1.2.3 Uninstall dynamic loaded driver

To uninstall the TPMC600 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 TPMC600 major device.

The implementation of the device information definition is done through a C structure, which is defined in the header file *tpmc600_info.h*.

This structure contains the following parameter:

PCIBusNumber	Contains the PCI bus number at which the supported device is connected. Valid bus numbers are in range from 0 to 255.
PCIDeviceNumber	Contains the device number (slot) at which the supported device is connected. Valid device numbers are in range from 0 to 31.

If both PCIBusNumber and PCIDeviceNumber are –1 then the driver will auto scan for supported devices. 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 there are more than one TPMC600 major devices.

A device information definition is unique for every TPMC600 major device. The file *tpmc600_info.c* on the distribution media contains two device information declarations, **tpmc600A** for the first major device and **tpmc600B** 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 a unique name, for example **tpmc600C**, **tpmc600D** and so on.

It is also necessary to modify the device and driver configuration file, respectively the configuration include file *tpmc600.cfg*.

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

```
TDRV600_INFO tpmc600A = {

     -1,           /*  Auto find the device 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 read this file and produces a new set of driver and device configuration tables and a corresponding nodetab.

To install the TPMC600 driver and devices into the LynxOS system, the configuration include file tpmc600.cfg must be included in the CONFIG.TBL (see also chapter 2.1.1.3).

The file tpmc600.cfg on the distribution disk contains the driver entry (*C:tpmc600:\....*) and two major device entries (*D:TPMC600 1:tpmc600A:: and D:TPMC600 2:tpmc600B::*).

If the driver should support more than one major device, the following entries for major devices must be enabled by removing the comment character (#). By copy and paste an existing major and minor entries and renaming the new entries, it is possible to add any number of additional TPMC600 devices.

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

```
#    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:tpmc600:tpmc600open:tpmc600close: \
::: \
::tpmc600ioctl: \
:tpmc600install:tpmc600uninstall
D:TPMC600 1:tpmc600A::
N:tpmc600a:0
D:TPMC600 2:tpmc600B::
N:tpmc600b:0
```

The configuration above creates the following nodes in the /dev directory.

```
/dev/tpmc600a /dev/tpmc600b
```

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

3 TPMC600 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 (TPMC600 device) named in ***path*** for reading and writing. The value of ***oflags*** indicates the intended use of the file. In case of a TPMC600 device ***oflags*** must be set to **O_RDWR** to open the file for both reading and writing.

The ***mode*** argument is required only when a file is created. Because a TPMC600 device already exists this argument is ignored.

EXAMPLE

```
int fd

fd = open ("/dev/tpmc600a", O_RDWR);
if (fd == -1)
{
    /* Handle error */
}
```

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;

result = close(fd);
if (result == -1)
{
     /* Handle error */
}
```

RETURNS

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

SEE ALSO

LynxOS System Call - close()

3.3 ioctl()

NAME

ioctl() – I/O device control

SYNOPSIS

#include <ioctl.h>
#include <tpmc600.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 ioctl codes are supported by the driver and are defined in *tpmc600.h*:

Symbol	**Meaning**
TPMC600_READ	Read state of the digital input lines
TPMC600_DEBENABLE	Configure and enable input debouncer
TPMC600_DEBDISABLE	Disable debouncer

See behind for more detailed information on each control code.

RETURNS

ioctl returns 0 if successful, or –1 on error.

On error, *errno* will contain a standard error code (see also LynxOS System Call – ioctl).

SEE ALSO

LynxOS System Call - ioctl().

3.3.1 TPMC600_READ

NAME

TPMC600_READ – 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_READ_BUFFER*) must be passed by the parameter *arg* to the device.

```
typedef struct
{
        int             mode;       /* read mode TPMC600_READ_xxx */
        unsigned int    mask;       /* mask for relevant event bits */
        int             timeout;    /* timeout in millseconds */
        unsigned int    value;      /* returned state of input lines */
} TPMC600_READ_BUFFER;
```

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_READ_NOW	The state of the input lines will be read immediately.
TPMC600_READ_HIGH_TR	The function will wait for a low to high transition on any of the input lines specified in *mask*
TPMC600_READ_LOW_TR	The function will wait for a high to low transition on any of the input lines specified in *mask*
TPMC600_READ_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_READ_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_READ_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 <tpmc600.h>

int                     fd;
int                     result;
TPMC600_READ_BUFFER     rdBuf;

/* --- read current input state (immediately) --- */
rdBuf.mode = TPMC600_READ_NOW;

result = ioctl(fd, TPMC600_READ, (char*)&rdBuf);
if (result >= 0)
{
     printf("INPUT: %08Xh\n", rdBuf.value);
}
else
{
     /* Read failed */
}

…

/* --- read input state after IN 4 switched from low to high --- */
rdBuf.mode    = TPMC600_READ_HIGH_TR;
rdBuf.mask    = (1 << 3);              /* set bit for IN 4 */
rdBuf.timeout = 10000;                 /* timeout after 10000 ticks */

result = ioctl(fd, TPMC600_READ, (char*)&rdBuf);
if (result >= 0)
{
     printf("INPUT: %08Xh\n", rdBuf.value);
}
else
{
     /* Read failed */
}
```

ERRORS

EINTR	The function was cancelled.
ETIMEDOUT	The maximum allowed time to finish the read request is exhausted.
EINVAL	An unsupported input parameter has been specified: - unknown read mode - specified input mask has specified no input line
EBUSY	The maximum number of active event jobs at the same time has been reached.

Other returned error codes are system error conditions.

3.3.2 TPMC600_DEBENABLE

NAME

TPMC600_DEBENABLE – Configure and enable input debouncer

DESCRIPTION

This function configures and enables the input debouncer. This allows ignoring short input transitions and disturbance on the input lines. A pointer to the callers specified debouncer time must be passed by the parameter *arg* to the device.

The value specifies the debouncer time in steps of 7µs. The minimum time is 7s (0) and the maximum time of ~440ms is specified by a debouncer value of 65535.

All signal changes shorter than the specified time will be filtered by the hardware.

Remember that also stable signal changes will not be recognized before the specified debouncer time has passed.

EXAMPLE

```
#include <tpmc600.h>

int             fd;
int             result;
int             debTime;

/* --- enable debouncer, (~10ms) --- */
debTime = 1428;             /* (10ms / 7µs */

result = ioctl(fd, TPMC600_DEBENABLE, (char*)&debTime);
if (result >= 0)
{
     /* Debouncer enabled */
}
else
{
     /* Enable debouncer failed */
}
```

3.3.3 TPMC600_DEBDISABLE

NAME

TPMC600_DEBDISABLE – Disable input debouncer

DESCRIPTION

This function disables the input debouncer. There are no parameters that must be passed by the parameter *arg* to the device.

EXAMPLE

```
#include <tpmc600.h>

int             fd;
int             result;
int             debTime;

/* --- disable debouncer --- */
result = ioctl(fd, TPMC600_DEBDISABLE, (char*)0);
if (result >= 0)
{
     /* Debouncer disabled */
}
else
{
     /* Disable debouncer failed */
}
```

4 Debugging and Diagnostic

If the driver will not work properly, please enable debug outputs by defining the symbols *DEBUG, DEBUG_TPMC,* and *DEBUG_PCI* in file tpmc600.c.

The debug output should appear on the console. If not, please check the symbol *KKPF_PORT* in *uparam.h*. This symbol should be configured to a valid COM port (e.g. *SKDB_COM1*).

The debug output displays the device information data for the current major device like this.

```
TPMC600: Device Driver Install
Bus = 0  Dev = 17  Func = 0
[00] = 02581498
[04] = 02800000
[08] = 11800001
[0C] = 00000000
[10] = 80003000
[14] = 00802001
[18] = 80004000
[1C] = 00000000
[20] = 00000000
[24] = 00000000
[28] = 00000000
[2C] = 000A1498
[30] = 00000000
[34] = 00000000
[38] = 00000000
[3C] = 0000010C

PCI Base Address 0 (PCI_RESID_BAR0)
70603000 : E0 FF FF 0F 00 00 00 00 00 00 00 00 00 00 00 00
70603010 : 00 00 00 00 01 00 00 00 00 00 00 00 00 00 00 00
70603020 : 00 00 00 00 00 00 00 00 80 78 B1 01 00 00 00 00
70603030 : 00 00 00 00 00 00 00 00 00 00 00 00 11 00 00 00

PCI Base Address 1 (PCI_RESID_BAR1)

PCI Base Address 2 (PCI_RESID_BAR2)
70604000 : 00000000 00000000 00000000 00000000
70604010 : 00000000 00000000 00000000 00000000
TPMC600: Found TPMC600-10 on Bus 0, Device 17
```

The debug output above is only an example. Debug output on other systems may be different for addresses and data in some locations.