The Embedded I/O Company

TEWS TECHNOLOGIES

TPMC600-SW-65

Windows 2000/XP Device Driver

32/16 Digital Inputs (24V)

Version 1.1.x

User Manual

Issue 1.1.0

January 2009

powerBridge Computer

Ehlbeek 15a
30938 Burgwedel
fon 05139-9980-0
fax 05139-9980-49

www.powerbridge.de
info@powerbridge.de

TEWS TECHNOLOGIES GmbH		**TEWS TECHNOLOGIES LLC**	
Am Bahnhof 7	Phone: +49 (0) 4101 4058 0	9190 Double Diamond Parkway,	Phone: +1 (775) 850 5830
25469 Halstenbek, Germany	Fax: +49 (0) 4101 4058 19	Suite 127, Reno, NV 89521, USA	Fax: +1 (775) 201 0347
www.tews.com	e-mail: info@tews.com	www.tews.com	e-mail: usasales@tews.com

TPMC600-SW-65

Windows 2000/XP 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.

©2003-2009 by TEWS TECHNOLOGIES GmbH

Issue	Description	Date
1.0.0	First Issue	August 28, 2003
1.1.0	General Revision	January 7, 2009

Table of Contents

1 Introduction

The TPMC600-SW-65 Windows 2000/XP device driver allows the operation of the TPMC600 Digital Input PMC conforming to the Windows 2000/XP I/O system specification. This includes a device-independent basic I/O interface with *open()*, *close()* and *ioctl()* functions.

The TPMC600-SW-65 device driver supports the following features:

- read the input port immediately without waiting for a specific input event
- read the input port after the following events occur
  - masked input bits match to the specified pattern
  - high-transition at the specified bit position
  - low-transition at the specified bit position
  - any transition (high or low) at the specified bit position
- configure & start and stop input debouncer

The TPMC600-SW-65 device driver supports the modules listed below:

TPMC600-10	32 digital input lines	(PMC)
TPMC600-11	16 digital input lines	(PMC)
TPMC600-20	32 digital input lines (Back-I/O)	(PMC)
TPMC600-21	16 digital input lines (Back-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-65\':

tpmc600.sys	TPMC600 Windows driver binary device driver source
tpmc600.inf	TPMC600 Windows installation file
tpmc600.h	TPMC600 include file for application
example/tpmc600exa.c	Example application (Microsoft Visual C)
TPMC600-SW-65-1.1.0.pdf	PDF copy of this manual
ChangeLog.txt	Release history
Release.txt	Release information

2.1 Software Installation

2.1.1 Windows 2000 / XP

This section describes how to install the TPMC600 Device Driver on a Windows 2000 / XP operating system.

After installing the TPMC600 card(s) and boot-up your system, Windows 2000 / XP setup will show a "***New hardware found***" dialog box.

1. The "***Upgrade Device Driver Wizard***" dialog box will appear on your screen.
   Click "***Next***" button to continue.
2. In the following dialog box, choose "***Search for a suitable driver for my device***".
   Click "***Next***" button to continue.
3. Insert the TPMC600 driver media; select "***Disk Drive***" in the dialog box.
   Click "***Next***" button to continue.
4. Now the driver wizard should find a suitable device driver on the media.
   Click "***Next***" button to continue.
5. Complete the upgrade device driver and click "***Finish***" to take all the changes effect.
6. Now copy all needed files (tpmc600.h, TPMC600-SW-65.pdf) to the desired target directories.

After successful installation the TPMC600 device driver will start immediately and creates devices (TPMC600_1, TPMC600_2 ...) for all recognized TPMC600 modules.

2.1.2 Confirming Windows 2000 / XP Installation

To confirm that the driver has been properly loaded in Windows 2000 / XP, perform the following steps:

1. From Windows 2000 / XP, open the "***Control Panel***" from "***My Computer***".
2. Click the "***System***" icon and choose the "***Hardware***" tab, and then click the "***Device Manager***" button.
3. Click the "**+**" in front of "***Other Devices***".
   The driver " ***TEWS TECHNOLOGIES TPMC600 (32/16 digital Input)***" should appear.

2.2 Change maximum number of event jobs

The TPMC600 device driver uses jobs, allocated during device startup, for event handling. There are just a limited number of jobs available, by default there are 10 jobs allocated.

If the default value is not suitable the configuration can be changed by modifying the registry, for instance with regedt32.

To change the maximum number of jobs the following value must be modified.

```
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\TPMC600\NumReqEntries
```

The size value must be greater than 0

After changing the value in the registry the device must be stopped and restarted or the system must be rebooted.

3 TPMC600 Device Driver Programming

The TPMC600-SW-65 Windows WDM device driver is a kernel mode device driver.

The standard file and device (I/O) functions (CreateFile, CloseHandle, and DeviceIoControl) provide the basic interface for opening and closing a device handle and for performing device I/O control operations.

All of these standard Win32 functions are described in detail in the Windows Platform SDK Documentation (Windows base services / Hardware / Device Input and Output).

For details refer to the Win32 Programmers Reference of your used programming tools (C++, Visual Basic etc.)

3.1 TPMC600 Files and I/O Functions

The following section doesn't contain a full description of the Win32 functions for interaction with the TPMC600 device driver. Only the required parameters are described in detail.

3.1.1 Opening a TPMC600 Device

Before you can perform any I/O the *TPMC600* device must be opened by invoking the **CreateFile** function. **CreateFile** returns a handle that can be used to access the *TPMC600* device.

```
HANDLE CreateFile(
    LPCTSTR lpFileName,                               // pointer to name of the file
    DWORD dwDesiredAccess,                            // access (read-write) mode
    DWORD dwShareMode,                                // share mode
    LPSECURITY_ATTRIBUTES lpSecurityAttributes,       // pointer to security attributes
    DWORD dwCreationDistribution,                     // how to create
    DWORD dwFlagsAndAttributes,                       // file attributes
    HANDLE hTemplateFile                              // handle to file with attributes to copy
)
```

lpFileName

Points to a null-terminated string, which specifies the name of the TPMC600 to open.
The *lpFileName* string should be of the form **\\.\TPMC600_*x*** to open the device *x*. The ending x is a one-based number. The first device found by the driver is \\.\TPMC600_1, the second \\.\TPMC600_2 and so on.

dwDesiredAccess

Specifies the type of the access to the TPMC600.
For the TPMC600 this parameter must be set to read-write access (GENERIC_READ | GENERIC_WRITE)

dwShareMode

Set of bit flags, that specifies how the object can be shared. Set to 0.

lpSecurityAttributes

Pointer to a security structure. Set to NULL for TPMC600 devices.

dwCreationDistribution

Specifies the action to take on files that exist, and which action to take when files do not exist. TPMC600 devices must be always opened *OPEN_EXISTING*.

dwFlagsAndAttributes

Specifies the file attributes and flags for the file. This value must be set to 0 (no overlapped I/O).

hTemplateFile

This value must be NULL for TPMC600 devices.

Return Value

If the function succeeds, the return value is an open handle to the specified TPMC600 device. If the function fails, the return value is INVALID_HANDLE_VALUE. To get extended error information, call ***GetLastError***.

Example

```
HANDLE    hDevice;

hDevice = CreateFile(
     “\\\\.\\TPMC600_1”,
     GENERIC_READ | GENERIC_WRITE,
     0,
     NULL,              // no security attrs
     OPEN_EXISTING,     // TPMC600 device always open existing
     0,                 // no overlapped I/O
     NULL
);

if (hDevice == INVALID_HANDLE_VALUE) {
     ErrorHandler("Could not open device"); // process error
}
```

See Also

CloseHandle(), Win32 documentation CreateFile()

3.1.2 Closing a TPMC600 Device

The **CloseHandle** function closes an open TPMC600 handle.

```
BOOL CloseHandle(
    HANDLE hDevice;                    // handle to a TPMC600 device to close
)
```

hDevice

Identifies an open TPMC600 handle.

Return Value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call ***GetLastError.***

Example

```
HANDLE    hDevice;

if( CloseHandle( hDevice ) ) {
     ErrorHandler( "Could not close device" ); // process error
}
```

See Also

CreateFile(), Win32 documentation CloseHandle()

3.1.3 TPMC600 Device I/O Control Functions

The **DeviceIoControl** function sends a control code directly to a specified device driver, causing the corresponding device to perform the specified operation.

```
BOOL DeviceIoControl(
        HANDLE hDevice,                   // handle to device of interest
        DWORD dwIoControlCode,            // control code of operation to perform
        LPVOID lpInBuffer,                // pointer to buffer to supply input data
        DWORD nInBufferSize,              // size of input buffer
        LPVOID lpOutBuffer,               // pointer to buffer to receive output data
        DWORD nOutBufferSize,             // size of output buffer
        LPDWORD lpBytesReturned,          // pointer to variable to receive output byte count
        LPOVERLAPPED lpOverlapped         // pointer to overlapped structure for asynchronous operation
);
```

hDevice

Handle to the TPMC600 that is to perform the operation.

dwIoControlCode

Specifies the control code for the operation. This value identifies the specific operation to be performed. The following values are defined in *tpmc600.h* :

Value	**Meaning**
IOCTL_TP600_READ	Read input port immediately
IOCTL_TP600_READ_EVENT	Read input port after specified event occur
IOCTL_TP600_DEBENABLE	Enable input debounce function
IOCTL_TP600_DEBDISABLE	Disable input debounce function

See behind for more detailed information on each control code.

lpInBuffer

Pointer to a buffer that contains the data required to perform the operation.

nInBufferSize

Specifies the size, in bytes, of the buffer pointed to by *lpInBuffer*.

lpOutBuffer

Pointer to a buffer that receives the operation's output data.

nOutBufferSize

Specifies the size, in bytes, of the buffer pointed to by *lpOutBuffer*.

lpBytesReturned

Pointer to a variable that receives the size, in bytes, of the data stored into the buffer pointed to by *lpOutBuffer*. A valid pointer is required.

lpOverlapped

Pointer to an *Overlapped* structure. This value must be set to NULL (no overlapped I/O).

Return Value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call ***GetLastError***.

Note. The TPMC600 driver returns always standard Win32 error codes on failure, please refer to the Windows Platform SDK Documentation for a detailed description of returned error codes.

See Also

Win32 documentation DeviceIoControl()

3.1.3.1 IOCTL_TP600_READ

This control function reads the input port of the TPMC600 associated with the open device handle.

The contents is returned in a unsigned long buffer pointed by *lpOutBuffer* . The buffer must be always an unsigned long type independent of the TPMC600 variant. The argument *nOutBufferSize* specifies the size (size of ULONG) of the read buffer.

For the TPMC600 variant 11/21 only the lower 16 bits are relevant

Example

```
#include <windows.h>
#include <winioctl.h>
#include "tpmc600.h"

HANDLE    hDevice;
BOOLEAN   success;
ULONG     NumBytes;
ULONG     PortData;

success = DeviceIoControl (
     hDevice,             //  TPMC600 handle
     IOCTL_TP600_READ,    //  control code
     NULL,
     0,
     &PortData,
     sizeof(PortData),
     &NumBytes,
     NULL                 //  not over lapped
);
if( success ) {
     printf("\nRead input port successful (input port = 0x%x)\n",
          PortData);
}
else {
     ErrorHandler ( "Device I/O control error" ); // process error
}
```

Error Codes

ERROR_INVALID_PARAMETER	This error is returned if the size of the read buffer is too small

See Also

Win32 documentation DeviceIoControl(), TPMC600 Hardware User Manual

3.1.3.2 IOCTL_TP600_READ_EVENT

The "event read function" reads the contents of the input port either immediately or after a specified event occur. Possible events are rising or falling edge or both, at a specified input bit or a pattern match of masked input bits.

Both parameter *lpInBuffer* and *lpOutBuffer* must pass a pointer to the read buffer (TP600_READ_BUFFER) to the device driver.

```
typedef struct {
        unsigned long       value;
        unsigned long       mode;
        unsigned long       mask;
        unsigned long       match;
        long                timeout;
} TP600_READ_BUFFER, *PTP600_READ_BUFFER;
```

value

Receives the contents of the input port

There is a delay between the specified event and the input value read that is based on the system and OS dependent interrupt latency.

mode

Specifies the "event" mode for this read request

TP600_NOW	The driver reads the input port and returns immediately to the caller. The parameter mask, match and timeout are not relevant in this mode. This mode is equal to the control function *IOCTL_TP600_READ*.
TP600_MATCH	The driver reads the input port if the masked input bits match to the specified pattern. The input mask must be specified in the parameter mask. A 1 value in the mask means than the input bit value "must-match" identically to the corresponding bit in the match parameter. It is not recommended using the match event; events may not be recognized because of interrupt latency.
TP600_HIGH_TR	The driver reads the input port, if a high-transition at the specified bit position occurs. A 1 value in mask specifies the bit position of the input port. If you specify more than one bit position the events are OR'ed. That means the read will be completed if a high-transition at least at one relevant bit position occur.
TP600_LOW_TR	The driver reads the input port, if a low-transition at the specified bit position occurs. A 1 value in mask specifies the bit position of the input port. If you specify more than one bit position the events are OR'ed. That means the read will be completed if a low-transition at least at one relevant bit position occur.
TP600_ANY_TR	The driver reads the input port, if a transition (high or low) at the specified bit position occurs. A 1 value in mask specifies the bit position of the input port. If you specify more than one bit position the events are OR'ed. That means the read will be completed if a transition at least at one relevant bit position occur.

mask

Specifies a bit mask. A 1 value marks the corresponding bit position as relevant.

match

Specifies a pattern that must match to the contents of the input port. Only the bit positions specified by *mask* must compare to the input port.

timeout

Specifies the amount of time (in seconds) the caller is willing to wait for the specified event to occur. A value of 0 means wait indefinitely.

Example

```
#include <windows.h>
#include <winioctl.h>
#include "tpmc600.h"

HANDLE              hDevice;
BOOLEAN             success;
ULONG               NumBytes;
TP600_READ_BUFFER   ReadBuf;

/*
**  Read input port immediately without waiting for any event
*/
ReadBuf.mode = TP600_NOW;

success = DeviceIoControl (
     hDevice,                     // TPMC600 handle
     IOCTL_TP600_READ_EVENT,
     &ReadBuf,                    // parameter for the driver
     sizeof(TP600_READ_BUFFER),
     &ReadBuf,                    // contains the read data
     sizeof(TP600_READ_BUFFER),
     &NumBytes,                   // size of returned ReadBuffer
     0
);
if( success ) {
     printf("Input port = 0x%x\n", ReadBuf.value);
}
else {
     ErrorHandler ( "Device I/O control error" ); // process error
}

…
```

```
…

/*
**  Read the input port after..
**   bit 0 = 0
**   bit 1 = 1
**   bit 6 = 0
**   bit 7 = 1
*/
ReadBuf.mode = TP600_MATCH;
ReadBuf.mask = 0x00C3;        // bit 0,1,6,7 are relevant
ReadBuf.match = 0x0082;
ReadBuf.timeout = 10;         // seconds

success = DeviceIoControl (
     hDevice,                 // TPMC600 handle
     IOCTL_TP600_READ_EVENT,
     &ReadBuf,                // parameter for the driver
     sizeof(TP600_READ_BUFFER),
     &ReadBuf,                // contains the read data
     sizeof(TP600_READ_BUFFER),
     &NumBytes,               // size of returned ReadBuffer
     0
);
if( success ) {
     printf("Input port = 0x%x\n", ReadBuf.value);
}
else {
     ErrorHandler ( "Device I/O control error" ); // process error
}


/*
**  Read the input port after a high-transition at bit 7
**  occured
*/
ReadBuf.mode = TP600_HIGH_TR;
ReadBuf.mask = 1<<7;          // high-transition at bit 7
ReadBuf.timeout = 10;         // seconds

…
```

```
…

success = DeviceIoControl (
     hDevice,                 // TPMC600 handle
     IOCTL_TP600_READ_EVENT,
     &ReadBuf,                // parameter for the driver
     sizeof(TP600_READ_BUFFER),
     &ReadBuf,                // contains the read data
     sizeof(TP600_READ_BUFFER),
     &NumBytes,               // size of returned ReadBuffer
     0
);
if( success ) {
     printf(“Input port = 0x%x\n”, ReadBuf.value);
}
else {
     ErrorHandler ( "Device I/O control error” ); // process error
}
```

Error Codes

ERROR_INVALID_PARAMETER	This error is returned if the size of the read buffer is too small or if the parameter mode contains an invalid value.
ERROR_NO_SYSTEM_RESOURCES	No more free entries in the drivers queue to handle concurrent event-controlled read requests. Increase the queue size.
ERROR_SEM_TIMEOUT	The requested event does not occur within the specified time (timeout).

All other returned error codes are system error conditions.

See Also

Win32 documentation DeviceIoControl(), TPMC600 Hardware User Manual

3.1.3.3 IOCTL_TP600_DEBENABLE

This control function enables the input debouncer function.

The new timer value is passed by an unsigned short variable, pointed by *lpInBuffer*, to the driver. The argument *nInBufferSize* specifies the size (size of USHORT) of the debouncer value.

See also TPMC600 Hardware User Manual – Debounce Time Register for counter calculation formulas.

Example

```
#include <windows.h>
#include <winioctl.h>
#include “tpmc600.h”

HANDLE    hDevice;
BOOLEAN   success;
ULONG     NumBytes;
USHORT    DebounceTime;

/*
**  Enable the debouncer with a debounce time of 1ms
*/
DebounceTime = 147;

success = DeviceIoControl (
     hDevice,                    //  TPMC600 handle
     IOCTL_TP600_DEBENABLE,      //  control code
     &DebounceTime,
     sizeof(DebounceTime),
     NULL,
     0,
     &NumBytes,
     NULL                        //  not overlapped
);
if( success ) {
     printf( "\nEnable output watchdog successful\n");
}
else {
     ErrorHandler ( "Device I/O control error” ); // process error
}
```

Error Codes

ERROR_INVALID_PARAMETER	This error is returned if the size of the timer value buffer is too small

See Also

Win32 documentation DeviceIoControl(), TPMC600 Hardware User Manual

3.1.3.4 IOCTL_TP600_DEBDISABLE

This control function disables the input debouncer function enabled by *IOCTL_TP600_DEBENABLE*.

There are no parameters required for this call.

Example

```
#include <windows.h>
#include <winioctl.h>
#include “tpmc600.h”

HANDLE    hDevice;
BOOLEAN   success;
ULONG     NumBytes;

success = DeviceIoControl (
     hDevice,                // TPMC600 handle
     IOCTL_TP600_DEBDISABLE, // control code
     NULL,
     0,
     NULL,
     0,
     &NumBytes,
     NULL                    // not over lapped
);
if( success ) {
      printf( "\nDisable output watchdog successful\n");
}
else {
     ErrorHandler ( "Device I/O control error” ); // process error
}
```

See Also

Win32 documentation DeviceIoControl(), TPMC600 Hardware User Manual