

TIP605-SW-62
Windows NT Device Driver
TIP605 – 16 Digital Inputs
Version 1.0

Reference Manual
Issue 1.0
October 20, 2000

TEWS DATENTECHNIK GmbH
Am Bahnhof 7
D-25469 Halstenbek
Germany
Tel.: +49 (0)4101 4058-0
Fax.: +49 (0)4101 4058-19
E-Mail: support@tews.com
Web: <http://tews.com>

TIP605-SW-62
TIP605 – 16 Digital Inputs
Windows NT Device Driver

This document contains information, which is proprietary to TEWS DATENTECHNIK GmbH. Any reproduction without written permission is forbidden.

TEWS DATENTECHNIK GmbH has made any effort to ensure that this manual is accurate and complete. However TEWS DATENTECHNIK GmbH reserves the right to change the product described in this document at any time without notice.

This product has been designed to operate with IndustryPack compatible carriers. Connection to incompatible hardware is likely to cause serious damage.

TEWS DATENTECHNIK GmbH is not liable for any damage arising out of the application or use of the device described herein.

2000 by TEWS DATENTECHNIK GmbH

Issue	Description	Date
1.0	First Issue	October 20, 2000

Table of Contents

1	INTRODUCTION	4
2	INSTALLATION.....	5
2.1	Software Installation	5
3	DRIVER CONFIGURATION	6
3.1	Driver Queue Configuration.....	6
4	TIP605 DEVICE DRIVER PROGRAMMING	7
4.1	TIP605 Files and I/O Functions.....	8
4.1.1	Opening a TIP605 Device	8
4.1.2	Closing a TIP605 Device.....	10
4.1.3	TIP605 Device I/O Control Functions	11
4.1.3.1	IOCTL_T605_READ.....	13
4.1.3.2	IOCTL_T605_SETTIME	17

1 Introduction

The TIP605-SW-62 Windows NT device driver is a kernel mode device driver which allows the operation of the TIP605 serial IP on an Intel or Intel-compatible x86 Windows NT 4.0 system in conjunction with a SBS IP-carrier board respective **SBS SDpack** device driver.

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

The TIP605 device driver includes the following functions:

- ☞ read the input port immediately without waiting for a specific input event
- ☞ read the input port if 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
- ☞ setup debounce timer register

2 Installation

The software is delivered on a 3½" HD diskette.

Following files are located on the diskette:

TIP605.sys	Windows NT driver binary
TIP605.h	Header file with IOCTL code definitions and driver specific data types
TIP605.inf	Windows NT installation script
TIP605-SW-62.pdf	Device Driver Manual
\Example\Example.c	Microsoft Visual C example application
\i386\checked*	Debug symbols and code

2.1 Software Installation

This section describes how to install the TIP605 Device Driver on a Windows NT 4.0 operating systems with Intel and Intel-compatible x86 CPU.

1. Plug the TIP605 card(s) on a SBS carrier board.
2. Switch on your system and log on as an Administrator.
3. Be sure that the SBS SDpack drivers (ipclass, pci40, ...) are installed and running.
4. Insert the TIP605 diskette into the floppy disk drive and copy all directories and files to the desired target directory.
5. Start Windows NT Explorer and look for a file called **TIP605.inf**
6. Right-click this file, select Install and follow the on-screen instructions.
7. Restart your computer when prompted.

The installation process copies the driver to the %SystemRoot%\SYSTEM32\DRIVERS directory and adds the following entries to the Registry:

HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\TIP605\..

and

HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\EventLog\System\TIP605

After successful installation the TIP605 device driver start automatically. The driver searches for TIP605 modules on SBS carrier boards, allocates necessary resources and creates devices (TIP605_1, TIP605_2, ...) for all found TIP605 modules.

3 Driver Configuration

3.1 Event Queue Configuration

After Installation of the TIP605 Device Driver the number concurrent event controlled read request is limited to 10.

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

To change the number of queue entries the following value must be modified.

HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\TIP605\NumReqEntries

Valid values are in range between 1..100

4 TIP605 Device Driver Programming

The TIP605-SW-62 Windows NT 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 resource 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.)

4.1 TIP605 Files and I/O Functions

The following section doesn't contain a full description of the Win32 functions for interaction with the TIP605 device driver. Only the required parameters are described in detail.

4.1.1 Opening a TIP605 Device

Before you can perform any I/O the *TIP605* device must be opened by invoking the **CreateFile** function. **CreateFile** returns a handle that can be used to access the *TIP605* device.

```
HANDLE CreateFile(
    LPCTSTR lpFileName,           // pointer to filename
    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
);
```

Parameters

lpFileName

Points to a null-terminated string that specifies the name of the TIP605 to open. The *lpFileName* string should be of the form \\.\TIP605_x to open the device x. The ending x is a one-based number. The first device found by the driver is \\.\TIP605_1, the second \\.\TIP605_2 and so on.

dwDesiredAccess

Specifies the type of access to the TIP605. For the TIP605 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 for read and write. Unimportant for TIP605, set to 0.

lpSecurityAttributes

Pointer to a security structure. Set to NULL for TIP605 devices.

dwCreationDistribution

Specifies which action to take on files that exist, and which action to take when files do not exist. TIP605 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 0 for TIP605 devices.

Return Value

If the function succeeds, the return value is an open handle to the specified TIP605 device. If the function fails, the return value is *INVALID_HANDLE_VALUE*. To get extended error information, call ***GetLastError***.

Example

```
HANDLE hDevice;  
  
hDevice = CreateFile(  
    "\\.\TIP605_1",  
    GENERIC_READ | GENERIC_WRITE,  
    0,  
    NULL, // no security attrs  
    OPEN_EXISTING, // TIP605 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()

4.1.2 Closing a TIP605 Device

The **CloseHandle** function closes an open TIP605 handle.

```
BOOL CloseHandle(  
    HANDLE hDevice;           // handle to a TIP605 device to close  
);
```

Parameters

hDevice

Identifies an open TIP605 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;  
  
hDevice = CreateFile(  
    "\\.\TIP605_1",  
    GENERIC_READ | GENERIC_WRITE,  
    0,  
    NULL,           // no security attrs  
    OPEN_EXISTING, // TIP605 device always open existing  
    0,             // no overlapped I/O  
    NULL  
);  
  
if( hDevice == INVALID_HANDLE_VALUE ) {  
    ErrorHandler( "Could not open device" ); // process error  
}  
  
/* ... do some device I/O ... */  
  
if( CloseHandle( hDevice ) ) {  
    ErrorHandler( "Could not close device" ); // process error  
}
```

See Also

CreateFile(), Win32 documentation **CloseHandle()**

4.1.3 TIP605 Device I/O Control Functions

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

```
BOOL DeviceloControl(  
    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  
);
```

Parameters

hDevice

Handle to the TIP605 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 *TIP605.h* :

Value	Meaning
<i>IOCTL_T605_READ</i>	Read the input port
<i>IOCTL_T605_SETTIME</i>	Setup debounce timer register

See behind for more detailed information on each control code.

Note

To use these TIP605 specific control codes the header file *TIP605.h* must be included in the application.

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***.

See Also

Win32 documentation DeviceIoControl()

4.1.3.1 IOCTL_T605_READ

The 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 (T605_READ_BUFFER) to the device driver.

The *T605_READ_BUFFER* structure has the following layout:

```
typedef struct {  
  
    unsigned short  value;  
    unsigned short  mode;  
    unsigned short  mask;  
    unsigned short  match;  
    long            timeout;  
  
} T605_READ_BUFFER, *PT605_READ_BUFFER;
```

value

Receives the contents of the input port.

mode

Specifies the “event” mode for this read request

T605_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.

T605_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 *mask* means that the input bit value “must-match” identically to the corresponding bit in the *match* parameter.

T605_HIGH_TR

The driver reads the input port if a high-transition at the specified bit position occur. 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 is completed if a high-transition at least at one relevant bit position occur.

T605_LOW_TR

The driver reads the input port if a low-transition at the specified bit position occur. 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 is completed if a low-transition at least at one relevant bit position occur.

T605_ANY_TR

The driver reads the input port if a transition (high or low) at the specified bit position occur. 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 is 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 "TIP605.h"

HANDLE hDevice;
BOOLEAN success;
ULONG NumBytes;
T605_READ_BUFFER ReadBuf;

/*
** Read input port immediately without waiting for any event
*/

ReadBuf.mode = T605_NOW;

success = DeviceIoControl (
    hDevice,                // TIP605 handle
    IOCTL_T605_READ,       // parameter for the driver
    &ReadBuf,               // contains the read data
    sizeof(T605_READ_BUFFER),
    &ReadBuf,              // size of returned ReadBuffer
    &NumBytes,
    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 = T605_MATCH;
ReadBuf.mask = 0x00C3;    // bit 0,1,6,7 are relevant
ReadBuf.match = 0x0082;
ReadBuf.timeout = 10;    // seconds
```

```

success = DeviceIoControl (
    hDevice, // TIP605 handle
    IOCTL_T605_READ,
    &ReadBuf, // parameter for the driver
    sizeof(T605_READ_BUFFER),
    &ReadBuf, // contains the read data
    sizeof(T605_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
** occurred
*/
ReadBuf.mode = T605_HIGH_TR;
ReadBuf.mask = 1<<7; // high-transition at bit 7
ReadBuf.timeout = 10; // seconds

success = DeviceIoControl (
    hDevice, // TIP605 handle
    IOCTL_T605_READ,
    &ReadBuf, // parameter for the driver
    sizeof(T605_READ_BUFFER),
    &ReadBuf, // contains the read data
    sizeof(T605_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

<i>ERROR_INVALID_PARAMETER</i>	This error is returned if the size of the read buffer is too small or if the parameter <i>mode</i> contains an invalid value.
<i>ERROR_NO_SYSTEM_RESOURCES</i>	No more free entries in the drivers queue to handle concurrent event-controlled read requests. Increase the queue size (see also 3.1).
<i>ERROR_SEM_TIMEOUT</i>	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()`, TIP605 Hardware User Manual

4.1.3.2 IOCTL_T605_SETTIME

This TIP605 control function writes a new value to the debounce time register. The parameter *lpInBuffer* passes a pointer to a unsigned char variable to the driver which contains the new value. See also *TIP605 Hardware Manuel – 4.8. Debounce Time Register* for more information.

Example

```
#include "TIP605.h"

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

//
// Setup debounce unit to maximum time (261 ms)
//

DebounceTime = 0xff;

success = DeviceIoControl (
    hDevice,                // TIP605 handle
    IOCTL_T605_SETTIME,    // setup new debounce time
    &DebounceTime,         // parameter for the driver
    sizeof(UCHAR),
    NULL,
    0,
    &NumBytes,            // unused
    0
);

if( !success ) {
    ErrorHandler ( "Device I/O control error" ); // process error
}
```

Error Codes

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

All other returned error codes are system error conditions.

See Also

Win32 documentation DeviceIoControl(), TIP605 Hardware User Manual