The Embedded I/O Company

TEWS TECHNOLOGIES

TIP630-SW-65

Windows 2000/XP Device Driver

Reconfigurable FPGA I/O IP

Version 1.0.x

User Manual

Issue 1.1

June 2004

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

TEWS TECHNOLOGIES LLC
1 E. Liberty Street, Sixth Floor
Reno, Nevada 89504 / USA
Phone: +1 (775) 686 6077
Fax: +1 (775) 686 6024
e-mail: usasales@tews.com
www.tews.com

TIP630-SW-65

Reconfigurable FPGA I/O IP

Windows 2000/XP Device Driver

Issue	Description	Date
1.0	First Issue	October 31, 2003
1.1	WinXP description added, Win98/Me description removed	June 14, 2004

Table of Content

1 Introduction

The TIP630-SW-65 Windows WDM (Windows Driver Model) device driver is a kernel mode driver which allows basic operations of the TIP630 on an Intel or Intel-compatible x86 Windows 2000/XP operating system.

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.

Because the TIP630 device driver is stacked on the TEWS TECHNOLOGIES IPAC carrier driver, it's necessary to install also the appropriate IPAC carrier driver. Please refer to the IPAC carrier driver user manual for further information.

The TIP630 device driver includes the following functions:

- reading and writing to and from ID, I/O and memory space
- programming the TIP630 FPGA Logic
- programming the clocks
- configuring the Carriers IPAC slot setting

To understand all features of this device driver, it is recommended to read the TIP630 User Manual.

2 Installation

Following files are located on the distribution disk:

tip630.sys	Device driver binary
tip630.h	Header file with IOCTL code definitions
tip630.inf	Installation script
TIP630-SW-65.pdf	This document
\Example\Example.c	Microsoft Visual C example application

2.1 Software Installation

The TIP630 Device Driver software assumes a correctly installed and active IPAC carrier driver.

2.1.1 Windows 2000/XP

This section describes how to install the TIP630 Device Driver on a Windows 2000/XP operating system.

After installing the TIP630 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 TIP630 driver disk; and select "***Disk Drive***" and/or "***CD-ROM***" in the dialog box.
   Click "***Next***" button to continue.
4. Now the driver wizard should find a suitable device driver on the diskette.
   Click "***Next***" button to continue.
5. If a window shows up announcing that the Windows Logo Test has failed, click **"continue install"** to continue the installation.
6. Complete the device driver installation and click "***Finish***" to take all the changes effect.
7. Now copy all needed files (tip630.h, TIP630-SW-65.pdf) to the desired target directories.

After successful installation the TIP630 device driver will start immediately and creates devices (TIP630_1, TIP630_2, ...) for all recognized TIP630 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 "***TIP630***" should appear.

3 Driver Configuration

3.1 Event Queue Configuration

After Installation of the TIP630 Device Driver the size of the TIP630s memory space is set to 0 Byte.

If the FPGA application supports memory accesses the default value must be changed by modifying the registry, for instance with regedt32 or regedit.

To change the memory size the following value must be modified.

HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\tip630\SizeMemSpace

The value must be set to the supported memory size.

After changing the memory size the driver must be restarted.

4 TIP630 Device Driver Programming

The TIP630-SW-65 Windows 2000/XP 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.)

4.1 TIP630 Files and I/O Functions

The following section doesn't contain a full description of the Win32 functions for interaction with the TIP630 device driver. Only the required parameters are described in detail.

4.1.1 Opening a TIP630 Device

Before you can perform any I/O the *TIP630* device must be opened by invoking the **CreateFile** function. **CreateFile** returns a handle that can be used to access the *TIP630* 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 TIP630 to open.
The *lpFileName* string should be of the form **\\.\TIP630_*x*** to open the device *x.* The ending x is a one-based number. The first device found by the driver is \\.\TIP630_1, the second \\.\TIP630_2 and so on.

dwDesiredAccess

Specifies the type of access to the TIP630. For the TIP630 this parameter must be set to read-write access (GENERIC_READ | GENERIC_WRITE).

dwShareMode

A set of bit flags that specifies how the object can be shared for read and write. Unimportant for TIP630, set to 0.

lpSecurityAttributes

Pointer to a security structure. Set to NULL for TIP630 devices.

dwCreationDistribution

Specifies which action to take on files that exist and which action to take when files that do not exist. TIP630 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 TIP630 devices.

Return Value

If the function succeeds, the return value is an open handle to the specified TIP630 device. If the function fails, the return value is INVALID_HANDLE_VALUE. To get extended error information, call **GetLastError**.

Example

```
HANDLE    hDevice;

hDevice = CreateFile(
     “\\\\.\\TIP630_1”,
     GENERIC_READ | GENERIC_WRITE,
     0,
     NULL,              // no security attrs
     OPEN_EXISTING,     // TIP630 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 TIP630 Device

The **CloseHandle** function closes an open TIP630 handle.

```
BOOL CloseHandle(
      HANDLE hDevice;                          // handle to a TIP630 device to close
);
```

Parameters

hDevice

Identifies an open TIP630 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(
     “\\\\.\\TIP630_1”,
     GENERIC_READ | GENERIC_WRITE,
     0,
     NULL,              // no security attrs
     OPEN_EXISTING,     // TIP630 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 TIP630 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
);
```

Parameters

hDevice

Handle to the TIP630 that is to perform the operation.

dwIoControlCode

Specifies the control code for an operation. This value identifies the specific operation to be performed. The following values are defined in *TIP630.h*:

Value	Meaning
TIP630_READ_UCHAR	Read a number of bytes (8-bit) from a specified address
TIP630_READ_USHORT	Read a number of words (16-bit) from a specified address
TIP630_READ_ULONG	Read a number of longwords (32-bit) from a specified address
TIP630_WRITE_UCHAR	Write a number of bytes (8-bit) to a specified address
TIP630_WRITE_USHORT	Write a number of words (16-bit) to a specified address
TIP630_WRITE_ULONG	Write a number of longwords (32-bit) to a specified address
TIP630_RECONFIGURE	Reconfigure IPClock and Bus-Width of the IPAC TIP630s slot.
TIP630_SET_CLOCK	Set values for clock setup
TIP630_PROGRAM_FPGA	Program the contents of the FPGA

See behind for more detailed information on each control code.

To use these TIP630 specific control codes the header file tip630.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 TIP630_READ_UCHAR

The read_uchar function reads bytewise (8-bit) the content of a specified area.

Both parameter *lpInBuffer* and *lpOutBuffer* must pass a pointer to the read buffer (*TIP630_IO_BUF*) to the device driver.

```
typedef struct _TIP630_IO_BUF
{
    UCHAR    space;                  // address space to read from
    ULONG    offset;                 // address offset in space
    ULONG    size;                   // number of uchar/ushort/ulong
    UCHAR    buffer[TIP630_MAXIOBUF];// pointer to buffer
} TIP630_IO_BUF, *PTIP630_IO_BUF;
```

space

This value specifies the IPAC space to be read from.

TIP630_IDSPACE	Access TIP630 ID-Space
TIP630_IOSPACE	Access TIP630 IO-Space
TIP630_MEMSPACE	Access TIP630 Memory-Space (if mapped)

offset

This value specifies the offset address in the specified space

size

This parameter specifies the number of bytes to be read

buffer

The read data will be returned in this buffer. The maximum size of the buffer is 128 byte.

Example

```
#include “tip630.h”

HANDLE hDevice;
BOOLEAN success;
ULONG NumBytes;
TIP630_IO_BUF RWBuf;
LONG i;

...
```

```
...

/*
**  Read buffer from ID-Space. 10 bytes from offset 0x10..0x19
*/
RWBuf.space = TIP630_IDSPACE;
RWBuf.offset = 0x10;
RWBuf.size = 10;
success = DeviceIoControl (
     hDevice,               // TIP630 handle
     TIP630_READ_UCHAR,
     &RWBuf,                // parameter for the driver
     sizeof(TIP630_IO_BUF),
     &RWBuf,                // contains the read data
     sizeof(TIP630_IO_BUF),
     &NumBytes,             // size of returned Buffer
     0
);
if( success ) {
     for (i = 0; i < RWBuf.size; i++) {
          printf("0x%02x\n", RWBuf.buffer[i]);
     }
}
else {
     ErrorHandler ( "Device I/O control error" );      // process error
}
```

Error Codes

STATUS_ACCESS_DENIED	Specified space is not mapped
STATUS_ADDRESS_NOT_ASSOCIATED	False space value specified
STATUS_INVALID_BUFFER_SIZE	Specified buffer size is invalid

All other returned error codes are system error conditions.

See Also

Win32 documentation DeviceIoControl(), TIP630 Hardware User Manual

4.1.3.2 TIP630_READ_USHORT

The read_ushort function reads wordwise (16-bit) the content of a specified area.

Both parameter *lpInBuffer* and *lpOutBuffer* must pass a pointer to the read buffer (*TIP630_IO_BUF*) to the device driver.

```
typedef struct _TIP630_IO_BUF
{
    UCHAR    space;                  // address space to read from
    ULONG    offset;                 // address offset in space
    ULONG    size;                   // number of uchar/ushort/ulong
    UCHAR    buffer[TIP630_MAXIOBUF]; // pointer to buffer
} TIP630_IO_BUF, *PTIP630_IO_BUF;
```

space

This value specifies the IPAC space to be read from.

TIP630_IDSPACE	Access TIP630 ID-Space
TIP630_IOSPACE	Access TIP630 IO-Space
TIP630_MEMSPACE	Access TIP630 Memory-Space (if mapped)

offset

This value specifies the offset address in the specified space

size

This parameter specifies the number of words to be read

buffer

The read data will be returned in this buffer. The maximum size of the buffer is 64 words.

Example

```
#include “tip630.h”

HANDLE hDevice;
BOOLEAN success;
ULONG NumBytes;
TIP630_IO_BUF RWBuf;
PUSHORT pusPtr;
LONG i;

...
```

```
...

/*
**  Read buffer from IO-Space. 8 words from offset 0x10..0x1F
*/
RWBuf.space = TIP630_IOSPACE;
RWBuf.offset = 0x10;
RWBuf.size = 8;
success = DeviceIoControl (
     hDevice,               // TIP630 handle
     TIP630_READ_USHORT,
     &RWBuf,                // parameter for the driver
     sizeof(TIP630_IO_BUF),
     &RWBuf,                // contains the read data
     sizeof(TIP630_IO_BUF),
     &NumBytes,             // size of returned Buffer
     0
);
if( success ) {
     pusPtr = (USHORT*)RWBuf.buffer;
     for (i = 0; i < RWBuf.size; i++) {
          printf("0x%04x\n", pusPtr[i]);
     }
}
else {
     ErrorHandler ( "Device I/O control error" );     // process error
}
```

Error Codes

STATUS_ACCESS_DENIED	Specified space is not mapped
STATUS_ADDRESS_NOT_ASSOCIATED	False space value specified
STATUS_INVALID_BUFFER_SIZE	Specified buffer size is invalid

All other returned error codes are system error conditions.

See Also

Win32 documentation DeviceIoControl(), TIP630 Hardware User Manual

4.1.3.3 TIP630_READ_ULONG

The read_ulong function reads longwordwise (32-bit) the content of a specified area.

Both parameter *lpInBuffer* and *lpOutBuffer* must pass a pointer to the read buffer (*TIP630_IO_BUF*) to the device driver.

```
typedef struct _TIP630_IO_BUF
{
    UCHAR    space;                  // address space to read from
    ULONG    offset;                 // address offset in space
    ULONG    size;                   // number of uchar/ushort/ulong
    UCHAR    buffer[TIP630_MAXIOBUF]; // pointer to buffer
} TIP630_IO_BUF, *PTIP630_IO_BUF;
```

space

This value specifies the IPAC space to be read from.

TIP630_IDSPACE	Access TIP630 ID-Space
TIP630_IOSPACE	Access TIP630 IO-Space
TIP630_MEMSPACE	Access TIP630 Memory-Space (if mapped)

offset

This value specifies the offset address in the specified space

size

This parameter specifies the number of longwords to be read

buffer

The read data will be returned in this buffer. The maximum size of the buffer is 32 longwords.

Example

```
#include “tip630.h”

HANDLE hDevice;
BOOLEAN success;
ULONG NumBytes;
TIP630_IO_BUF RWBuf;
PULONG pulPtr;
LONG i;

...
```

```
...

/*
**  Read buffer from Memory-Space. 16 longwords from offset 0x00..0x3F
*/
RWBuf.space = TIP630_MEMSPACE;
RWBuf.offset = 0x00;
RWBuf.size = 0x10;
success = DeviceIoControl (
     hDevice,               // TIP630 handle
     TIP630_READ_UCHAR,
     &RWBuf,                // parameter for the driver
     sizeof(TIP630_IO_BUF),
     &RWBuf,                // contains the read data
     sizeof(TIP630_IO_BUF),
     &NumBytes,             // size of returned Buffer
     0
);
if( success ) {
     pulPtr = (ULONG*)RWBuf.buffer;
     for (i = 0; i < RWBuf.size; i++) {
          printf("0x%04lx\n", pilPtr[i]);
     }
}
else {
     ErrorHandler ( "Device I/O control error" );     // process error
}
```

Error Codes

STATUS_ACCESS_DENIED	Specified space is not mapped
STATUS_ADDRESS_NOT_ASSOCIATED	False space value specified
STATUS_INVALID_BUFFER_SIZE	Specified buffer size is invalid

All other returned error codes are system error conditions.

See Also

Win32 documentation DeviceIoControl(), TIP630 Hardware User Manual

4.1.3.4 TIP630_WRITE_UCHAR

The write_uchar function writes data from a buffer bytewise (8-bit) to a specified area.

Both parameter *lpInBuffer* and *lpOutBuffer* must pass a pointer to the write buffer (*TIP630_IO_BUF*) to the device driver.

```
typedef struct _TIP630_IO_BUF
{
    UCHAR    space;                  // address space to read from
    ULONG    offset;                 // address offset in space
    ULONG    size;                   // number of uchar/ushort/ulong
    UCHAR    buffer[TIP630_MAXIOBUF]; // pointer to buffer
} TIP630_IO_BUF, *PTIP630_IO_BUF;
```

space

This value specifies the IPAC space to write to.

TIP630_IDSPACE	Access TIP630 ID-Space
TIP630_IOSPACE	Access TIP630 IO-Space
TIP630_MEMSPACE	Access TIP630 Memory-Space (if mapped)

offset

This value specifies the offset address in the specified space

size

This parameter specifies the number of bytes to write

buffer

The write data must be specified in this buffer. The maximum size of the buffer is 128 byte.

Example

```
#include “tip630.h”

HANDLE hDevice;
BOOLEAN success;
ULONG NumBytes;
TIP630_IO_BUF RWBuf;

...
```

```
...

/*
**  Write buffer to IO-Space. 4 bytes from offset 0x10..0x13
*/
RWBuf.space = TIP630_IOSPACE;
RWBuf.offset = 0x10;
RWBuf.size = 4;
RWBuf.buffer[0] = 0x01;
RWBuf.buffer[1] = 0x02;
RWBuf.buffer[2] = 0x03;
RWBuf.buffer[3] = 0x04;
success = DeviceIoControl (
     hDevice,               // TIP630 handle
     TIP630_WRITE_UCHAR,
     &RWBuf,                // parameter for the driver
     sizeof(TIP630_IO_BUF),
     &RWBuf,
     sizeof(TIP630_IO_BUF),
     &NumBytes,             // size of returned Buffer
     0
);
if( success ) {
     /* Write succeeded */
}
else {
     ErrorHandler ( "Device I/O control error” );      // process error
}
```

Error Codes

Error Codes

STATUS_ACCESS_DENIED	Specified space is not mapped
STATUS_ADDRESS_NOT_ASSOCIATED	False space value specified
STATUS_INVALID_BUFFER_SIZE	Specified buffer size is invalid

All other returned error codes are system error conditions.

See Also

Win32 documentation DeviceIoControl(), TIP630 Hardware User Manual

4.1.3.5 TIP630_WRITE_USHORT

The write_ushort function writes data from a buffer wordwise (16-bit) to a specified area.

Both parameter *lpInBuffer* and *lpOutBuffer* must pass a pointer to the write buffer (*TIP630_IO_BUF*) to the device driver.

```
typedef struct _TIP630_IO_BUF
{
    UCHAR    space;                  // address space to read from
    ULONG    offset;                 // address offset in space
    ULONG    size;                   // number of uchar/ushort/ulong
    UCHAR    buffer[TIP630_MAXIOBUF];// pointer to buffer
} TIP630_IO_BUF, *PTIP630_IO_BUF;
```

space

This value specifies the IPAC space to write to.

TIP630_IDSPACE	Access TIP630 ID-Space
TIP630_IOSPACE	Access TIP630 IO-Space
TIP630_MEMSPACE	Access TIP630 Memory-Space (if mapped)

offset

This value specifies the offset address in the specified space

size

This parameter specifies the number of words to write

buffer

The write data must be specified in this buffer. The maximum size of the buffer is 64 words.

Example

```
#include “tip630.h”

HANDLE hDevice;
BOOLEAN success;
ULONG NumBytes;
TIP630_IO_BUF RWBuf;
PUSHORT pusPtr;

...
```

```
...

/*
**  Write buffer to IO-Space. 4 words from offset 0x10..0x17
*/
RWBuf.space = TIP630_IOSPACE;
RWBuf.offset = 0x10;
RWBuf.size = 4;
pusPtr = (PUSHORT)RWBuf.buffer;
pusPtr [0] = 0x0102;
pusPtr [1] = 0x0304;
pusPtr [2] = 0x0506;
pusPtr [3] = 0x0708;
success = DeviceIoControl (
     hDevice,                // TIP630 handle
     TIP630_WRITE_USHORT,
     &RWBuf,                 // parameter for the driver
     sizeof(TIP630_IO_BUF),
     &RWBuf,
     sizeof(TIP630_IO_BUF),
     &NumBytes,              // size of returned Buffer
     0
);
if( success ) {
     /* Write succeeded */
}
else {
     ErrorHandler ( "Device I/O control error” );      // process error
}
```

Error Codes

STATUS_ACCESS_DENIED	Specified space is not mapped
STATUS_ADDRESS_NOT_ASSOCIATED	False space value specified
STATUS_INVALID_BUFFER_SIZE	Specified buffer size is invalid

All other returned error codes are system error conditions.

See Also

Win32 documentation DeviceIoControl(), TIP630 Hardware User Manual

4.1.3.6 TIP630_WRITE_ULONG

The write_ulong function writes data from a buffer longwordwise (32-bit) to a specified area.

Both parameter *lpInBuffer* and *lpOutBuffer* must pass a pointer to the write buffer (*TIP630_IO_BUF*) to the device driver.

```
typedef struct _TIP630_IO_BUF
{
    UCHAR    space;                  // address space to read from
    ULONG    offset;                 // address offset in space
    ULONG    size;                   // number of uchar/ushort/ulong
    UCHAR    buffer[TIP630_MAXIOBUF]; // pointer to buffer
} TIP630_IO_BUF, *PTIP630_IO_BUF;
```

space

This value specifies the IPAC space to write to.

TIP630_IDSPACE	Access TIP630 ID-Space
TIP630_IOSPACE	Access TIP630 IO-Space
TIP630_MEMSPACE	Access TIP630 Memory-Space (if mapped)

offset

This value specifies the offset address in the specified space

size

This parameter specifies the number of longwords to write

buffer

The write data must be specified in this buffer. The maximum size of the buffer is 32 longwords.

Example

```
#include “tip630.h”

HANDLE hDevice;
BOOLEAN success;
ULONG NumBytes;
TIP630_IO_BUF RWBuf;
PULONG pulPtr;

...
```

```
...

/*
**  Write buffer to IO-Space. 4 longwords from offset 0x10..0x1F
*/
RWBuf.space = TIP630_IOSPACE;
RWBuf.offset = 0x10;
RWBuf.size = 4;
pulPtr = (PULONG)RWBuf.buffer;
pulPtr [0] = 0x01020304;
pulPtr [1] = 0x05060708;
pulPtr [2] = 0x090a0b0c;
pulPtr [3] = 0x0d0e0f00;
success = DeviceIoControl (
     hDevice,                // TIP630 handle
     TIP630_WRITE_ULONG,
     &RWBuf,                 // parameter for the driver
     sizeof(TIP630_IO_BUF),
     &RWBuf,
     sizeof(TIP630_IO_BUF),
     &NumBytes,              // size of returned Buffer
     0
);
if( success ) {
     /* Write succeeded */
}
else {
     ErrorHandler ( "Device I/O control error” );      // process error
}
```

Error Codes

STATUS_ACCESS_DENIED	Specified space is not mapped
STATUS_ADDRESS_NOT_ASSOCIATED	False space value specified
STATUS_INVALID_BUFFER_SIZE	Specified buffer size is invalid

All other returned error codes are system error conditions.

See Also

Win32 documentation DeviceIoControl(), TIP630 Hardware User Manual

4.1.3.7 TIP630_RECONFIGURE

The reconfigure function reconfigures the IPAC port the TIP630 is mounted to. This may be necessary after loading a new FPGA code or after startup, if the TIP630 should not use 8MHz IP-clock and 16-bit IP-Bus width.

Both parameter *lpInBuffer* and *lpOutBuffer* must pass a pointer to the configuration buffer (*TIP630_RECONFIG_BUF*) to the device driver.

```
typedef struct _TIP630_RECONFIG_BUF
{
    BOOLEAN         busSize8Bit;    // TRUE  - 8-bit Bus
                                    // FALSE - 16-bit Bus [default]
    BOOLEAN         ipClock32MHz;   // TRUE  - 32MHz IP-Clock
                                    // FALSE - 8MHz IP-Clock [default]
} TIP630_RECONFIG_BUF, *PTIP630_RECONFIG_BUF;
```

busSize8Bit

This value specifies the IP-Bus size.

TRUE	8-bit Bus
FALSE	16-bit Bus (default)

ipClock32MHz

This value specifies the IP-Clock speed.

TRUE	32 MHz
FALSE	8 MHz (default)

Example

```
#include “tip630.h”

HANDLE hDevice;
BOOLEAN success;
ULONG NumBytes;
TIP630_RECONFIG_BUF configBuf;

/*
**  Configure IPAC slot for 16-Bit Bus and 8MHz IP clock
*/
configBuf. busSize8Bit = FALSE;
configBuf. ipClock32MHz = FALSE;

...
```

```
...

success = DeviceIoControl (
     hDevice,                // TIP630 handle
     TIP630_RECONFIGURE,
     &configBuf,             // parameter for the driver
     sizeof(TIP630_RECONFIG_BUF),
     &configBuf,
     sizeof(TIP630_RECONFIG_BUF),
     &NumBytes,              // size of returned Buffer
     0
);
if( success ) {
     /* Configuring succeeded */
}
else {
     ErrorHandler ( "Device I/O control error” );      // process error
}
```

Error Codes

All returned error codes are system error conditions.

See Also

Win32 documentation DeviceIoControl(), TIP630 Hardware User Manual

4.1.3.8 TIP630_SET_CLOCK

The set_clock function configures both local clocks of the TIP630. The needed data should be taken from a tool called ‘cyberclocks’ which calculates the register settings for the Cypress clock devices.

Both parameter *lpInBuffer* and *lpOutBuffer* must pass a pointer to the clock buffer (*T630_CLOCK_PARAM*) to the device driver.

```
typedef struct _T630_CLOCK_PARAM
{
    UCHAR       ClkOE;      // 2bit relevant
    UCHAR       Div1Src;    // 1bit
    UCHAR       Div1N;      // 7bit
    UCHAR       XDrv;       // 2bit
    UCHAR       CapLoad;    // 8bit
    UCHAR       Pump;       // 3bit
    USHORT      PBcount;    // 10bit
    UCHAR       POcount;    // 1bit
    UCHAR       Qcount;     // 7bit
    UCHAR       ClkASrc;    // 3bit
    UCHAR       ClkBSrc;    // 3bit
    UCHAR       Div2Src;    // 1bit
    UCHAR       Div2N;      // 7bit
} T630_CLOCK_PARAM, *PT630_CLOCK_PARAM;
```

The parameters are all values defined by Cypress and the calculation tool.

Example

```
#include “tip630.h”

HANDLE hDevice;
BOOLEAN success;
ULONG NumBytes;
T630_CLOCK_PARAM clockBuf =
     {0x03, 0x00, 0x32, 0x01, 0x00, 0x01, 0x0008, 0x01,
      0x06, 0x01, 0x04, 0x00, 0x19};

...
```

```
...

/*
**  Configure clocks (2MHz/4MHz)
*/
success = DeviceIoControl (
     hDevice,               // TIP630 handle
     TIP630_SET_CLOCK,
     &clockBuf,             // parameter for the driver
     sizeof(T630_CLOCK_PARAM),
     &clockBuf,
     sizeof(T630_CLOCK_PARAM),
     &NumBytes,             // size of returned Buffer
     0
);
if( success )
     /* Setting succeeded */
}
else {
     ErrorHandler ( "Device I/O control error” );      // process error
}
```

Error Codes

STATUS_ACCESS_DENIED	Programming the clock failed

All other returned error codes are system error conditions.

See Also

Win32 documentation DeviceIoControl(), TIP630 Hardware User Manual

4.1.3.9 TIP630_PROGRAM_FPGA

The program_FPGA function programs the TIP630s FPGA logic.

Both parameter *lpInBuffer* and *lpOutBuffer* must pass a pointer of the character array with FPGA data to the device driver.

The buffer must be an array of UCHAR that contains the FPGA data. The length of the buffer is fixed to *SIZE_OF_FPGA_DATA* (0x28C44 bytes).

Example

```
#include “tip630.h”

extern PUCHAR FPGAbuf;

HANDLE hDevice;
BOOLEAN success;
ULONG NumBytes;

/*
**  Program FPGA with extenal data
*/
success = DeviceIoControl (
     hDevice,                // TIP630 handle
     TIP630_PROGRAM_FPGA,
     FPGAbuf,                // parameter for the driver
     sizeof(FPGAbuf),
     FPGAbuf,
     sizeof(FPGAbuf),
     &NumBytes,              // size of returned Buffer
     0
);
if( success ) {
     /* Programming succeeded */
}
else {
     ErrorHandler ( "Device I/O control error” );      // process error
}
```

Error Codes

STATUS_ACCESS_DENIED	Programming the FPGA failed

All other returned error codes are system error conditions.

See Also

Win32 documentation DeviceIoControl(), TIP630 Hardware User Manual