

---

# TIP630-SW-95

## QNX-Neutrino Device Driver

Reconfigurable FPGA Digital I/O

Version 1.0.x

## User Manual

Issue 1.0.0

January 2005

**TIP630-SW-95**

Reconfigurable FPGA Digital I/O

QNX-Neutrino Device Driver

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.

©2005 by TEWS TECHNOLOGIES GmbH

<b>Issue</b>	<b>Description</b>	<b>Date</b>
1.0.0	First Issue	February 01, 2005

## Table of Content

<b>1</b>	<b>INTRODUCTION.....</b>	<b>4</b>
<b>2</b>	<b>INSTALLATION.....</b>	<b>5</b>
	2.1 Build the device driver .....	5
	2.2 Build the example application .....	5
	2.3 Start the driver process.....	6
<b>3</b>	<b>DEVICE INPUT/OUTPUT FUNCTIONS .....</b>	<b>7</b>
	3.1 open() .....	7
	3.2 close().....	8
	3.3 devctl() .....	9
	3.3.1 DCMD_TIP630_READ_UCHAR.....	11
	3.3.2 DCMD_TIP630_READ_USHORT .....	13
	3.3.3 DCMD_TIP630_READ_ULONG.....	15
	3.3.4 DCMD_TIP630_WRITE_UCHAR .....	17
	3.3.5 DCMD_TIP630_WRITE_USHORT.....	19
	3.3.6 DCMD_TIP630_WRITE_ULONG .....	21
	3.3.7 DCMD_TIP630_RECONFIGURE .....	23
	3.3.8 DCMD_TIP630_SET_CLOCK .....	25
	3.3.9 DCMD_TIP630_PROGRAM_FPGA .....	27
	3.3.10 DCMD_TIP630_MEMSPACE_MAP.....	29
	3.3.11 DCMD_TIP630_MEMSPACE_UNMAP.....	30

---

# **1 Introduction**

The TIP630-SW-95 QNX-Neutrino device driver allows the operation of a TIP630 Reconfigurable FPGA Digital I/O IP on QNX-Neutrino operating systems and requires the TEWS TECHNOLOGIES QNX-Neutrino IPAC Carrier Driver Software (CARRIER-SW-95).

The TIP630 device driver is basically implemented as a user-installable Resource Manager. The standard file (I/O) functions (*open*, *close* and *devctl*) provide the basic interface for opening and closing a file descriptor and for performing device I/O and control operations.

Supported features:

- 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 very important to read the *Hardware User Manual* of the TIP630.

## 2 Installation

The TIP630-SW-95 directory on the distribution media contains the following files:

TIP630-SW-95.pdf	This manual in PDF format
TIP630-SW-95.tar.gz	Archive with driver source code

The archive TIP630-SW-95.tar.gz contains the following files and directories:

TIP630/driver/tip630.c	Driver source code
TIP630/driver/tip630.h	Driver interface definitions and data structures
TIP630/driver/tip630def.h	Device driver include
TIP630/driver/Makefile	Script for make utility to build and install the driver
TIP630/example/example.c	Example application
TIP630/example/*	additional files and directories for example application

In order to perform an installation, first login as *root* and copy the TAR archive to the */usr/src* directory and then extract all files and directories.

**Before building a new device driver, the TEWS TECHNOLOGIES IPAC Carrier Driver must be installed properly, because this driver includes the header files *ipac\_\*.h*, which are part of the IPAC Carrier Driver distribution. Please refer to the IPAC Carrier Driver user manual in the directory path */CARRIER-SW-95* on the corresponding distribution media.**

It is absolute important to extract the TIP630-SW-95.tar in the */usr/src* directory. Otherwise the automatic build with make will fail.

### 2.1 Build the device driver

Change to the */usr/src/TIP630/driver* directory

Execute the Makefile

```
# make install
```

After successful completion the driver library will be installed in the */lib/dll* directory.

**Make sure that the IPAC Carrier Driver configuration file “*/etc/IPAC\_CARRIER/ipac\_driver.txt*” contains the following entry:**

<b>tip630.so</b>	<b>b3</b>	<b>38</b>	<b>1</b>	<b>Reconfigurable FPGA Digital I/O</b>
------------------	-----------	-----------	----------	--

### 2.2 Build the example application

Change to the */usr/src/tip630/example* directory

Execute the Makefile

```
# make install
```

After successful completion the example binary (*tip630exam*) will be installed in the */bin* directory.

---

## 2.3 Start the driver process

To start the TIP630 Resource Manager (Device Driver) you only have to start the TEWS TECHNOLOGIES IPAC Carrier Driver. The Carrier Driver automatically detects installed TEWS IPs and dynamically loads the concerning modul(s).

The TIP630 Resource Manager registers a device for each TIP630 in the QNX-Neutrino's pathname space under following name, where n specifies the used IPAC slot number (please refer to the IPAC Carrier Driver Manual):

```
/dev/tip630_n
```

This pathname must be used in the application program to open a path to the desired TIP630 device.

For debugging purposes you can start the IPAC Carrier Driver with the `-V` (verbose) option. Now the Resource Manager will print versatile information about TIP630 configuration and command execution on the terminal window. For further details about debugging see IPAC Carrier Driver Manual.

## **3 Device Input/Output functions**

This chapter describes the interface to the device driver I/O system.

### **3.1 open()**

#### **NAME**

open() - open a file descriptor

#### **SYNOPSIS**

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
```

```
int open(const char *pathname, int flags)
```

#### **DESCRIPTION**

The open function creates and returns a new file descriptor for the TIP630 named by pathname.

The flags argument controls how the file is to be opened. TIP630 devices must be opened O\_RDWR.

#### **EXAMPLE**

```
int fd;

fd = open("/dev/tip630_0", O_RDWR);
```

#### **RETURNS**

The normal return value from open is a non-negative integer file descriptor. In the case of an error, a value of -1 is returned. The global variable *errno* contains the detailed error code.

#### **ERRORS**

Returns only Neutrino specific error codes, see Neutrino Library Reference.

#### **SEE ALSO**

Library Reference - open()

## 3.2 close()

### NAME

close() – close a file descriptor

### SYNOPSIS

```
#include <unistd.h>
```

```
int close (int filedes)
```

### DESCRIPTION

The **close** function closes the file descriptor *filedes*.

### EXAMPLE

```
int fd;

...

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

...
```

### RETURNS

The normal return value from close is 0. In the case of an error, a value of -1 is returned. The global variable *errno* contains the detailed error code.

### ERRORS

Returns only Neutrino specific error code, see Neutrino Library Reference.

### SEE ALSO

Library Reference - close()

## 3.3 devctl()

### NAME

devctl() – device control functions

### SYNOPSIS

```
#include <sys/types.h>
#include <unistd.h>
#include <devctl.h>

int devctl
(
    int    filedes,
    int    dcmd,
    void   *data_ptr,
    size_t n_bytes,
    int    dev_info_ptr
);
```

### DESCRIPTION

The **devctl** function sends a control code directly to a device, specified by *filedes*, causing the corresponding device to perform the requested operation.

The argument *dcmd* specifies the control code for the operation.

The arguments *data\_ptr* and *n\_bytes* depends on the command and will be described for each command in detail later in this chapter. Usually *data\_ptr* points to a buffer that passes data between the user task and the driver and *n\_bytes* defines the size of this buffer.

The argument *dev\_info\_ptr* is unused for the TIP630 driver and should be set to NULL.

The following devctl command codes are defined in *TIP630.h*:

<i>Value</i>	<i>Meaning</i>
<i>DCMD_TIP630_READ_UCHAR</i>	Read a number of bytes (8-bit) from a specified address
<i>DCMD_TIP630_READ_USHORT</i>	Read a number of words (16-bit) from a specified address
<i>DCMD_TIP630_READ_ULONG</i>	Read a number of longwords (32-bit) from a specified address
<i>DCMD_TIP630_WRITE_UCHAR</i>	Write a number of bytes (8-bit) to a specified address
<i>DCMD_TIP630_WRITE_USHORT</i>	Write a number of words (16-bit) to a specified address
<i>DCMD_TIP630_WRITE_ULONG</i>	Write a number of longwords (32-bit) to a specified address
<i>DCMD_TIP630_RECONFIGURE</i>	Reconfigure IPClock and Bus-Width of the IPAC TIP630s slot.
<i>DCMD_TIP630_SET_CLOCK</i>	Set values for clock setup
<i>DCMD_TIP630_PROGRAM_FPGA</i>	Program the contents of the FPGA
<i>DCMD_TIP630_MEMSPACE_MAP</i>	Map the TIP630's memory space
<i>DCMD_TIP630_MEMSPACE_UNMAP</i>	Unmap the TIP30's memory space

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**

## RETURNS

On success, EOK is returned. In the case of an error, the appropriate error code is returned by the function (not in errno!).

## ERRORS

ENOTTY	Inappropriate I/O control operation. This error code is returned if the requested devctl function is unknown. Please check the argument <i>dcmd</i> .
--------	---

Other function dependant error codes will be described for each devctl code separately. Note, the TIP630 driver always returns standard QNX error codes.

## SEE ALSO

Library Reference - devctl()

### 3.3.1 DCMD\_TIP630\_READ\_UCHAR

#### NAME

DCMD\_TIP630\_READ\_UCHAR - Read a number of bytes (8-bit) from a specified address

#### DESCRIPTION

This devctl function reads a number of bytes (8-bit) from a specified address. A pointer to the caller's buffer (*TIP630\_IO\_BUF*) and the size of this structure is passed by the parameters *data\_ptr* and *n\_bytes* to the device driver.

The *TIP630\_IO\_BUF* structure has the following layout:

```
typedef struct
{
    unsigned char    space;           /* address space to read from */
    unsigned char    offset;         /* address offset in space */
    unsigned char    size;           /* number of uchar/ushort/ulong */
    unsigned char    buffer[TIP630_MAXIOBUF]; /* buffer where data is stored */
} 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"

int fd;
int result;
TIP630_IO_BUF    ioBuf;
unsigned char    *pucPtr;

...

/*
**  Read buffer from ID-Space. 10 bytes from offset 0x10..0x19
*/
ioBuf.space      = TIP630_IDSPACE;
ioBuf.offset     = 0x10;
ioBuf.size       = 10;

result = devctl(    fd,
                   DCMD_TIP630_READ_UCHAR,
                   &ioBuf,
                   sizeof(TIP630_IO_BUF),
                   NULL);

if (result == EOK)
{
    pucPtr = (unsigned char*)ioBuf.buffer;
    for (i = 0; i < ioBuf.size; i++) {
        printf("0x%02x\n", pucPtr[i]);
    }
}
```

## ERRORS

EINVAL  
ENOMEM

The specified parameters are invalid.  
The supplied buffer is too small.

## SEE ALSO

Library Reference - devctl()

### 3.3.2 DCMD\_TIP630\_READ\_USHORT

#### NAME

DCMD\_TIP630\_READ\_USHORT - Read a number of words (16-bit) from a specified address

#### DESCRIPTION

This devctl function reads a number of words (16-bit) from a specified address. A pointer to the caller's buffer (*TIP630\_IO\_BUF*) and the size of this structure is passed by the parameters *data\_ptr* and *n\_bytes* to the device driver.

The *TIP630\_IO\_BUF* structure has the following layout:

```
typedef struct
{
    unsigned char    space;           /* address space to read from */
    unsigned char    offset;         /* address offset in space */
    unsigned char    size;           /* number of uchar/ushort/ulong */
    unsigned char    buffer[TIP630_MAXIOBUF]; /* buffer where data is stored */
} 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"

int fd;
int result;
TIP630_IO_BUF      ioBuf;
unsigned short      *pusPtr;

...

/*
**  Read buffer from ID-Space. 10 words from offset 0x10..0x19
*/
ioBuf.space      = TIP630_IDSPACE;
ioBuf.offset     = 0x10;
ioBuf.size       = 10;

result = devctl(   fd,
                  DCMD_TIP630_READ_USHORT,
                  &ioBuf,
                  sizeof(TIP630_IO_BUF),
                  NULL);

if (result == EOK)
{
    pusPtr = (unsigned short*)ioBuf.buffer;
    for (i = 0; i < ioBuf.size; i++) {
        printf("0x%04x\n", pusPtr[i]);
    }
}
```

## ERRORS

EINVAL  
ENOMEM

The specified parameters are invalid.  
The supplied buffer is too small.

## SEE ALSO

Library Reference - devctl()

### 3.3.3 DCMD\_TIP630\_READ\_ULONG

#### NAME

DCMD\_TIP630\_READ\_ULONG - Read a number of longwords (32-bit) from a specified address

#### DESCRIPTION

This devctl function reads a number of longwords (32-bit) from a specified address. A pointer to the caller's buffer (*TIP630\_IO\_BUF*) and the size of this structure is passed by the parameters *data\_ptr* and *n\_bytes* to the device driver.

The *TIP630\_IO\_BUF* structure has the following layout:

```
typedef struct
{
    unsigned char    space;           /* address space to read from */
    unsigned char    offset;         /* address offset in space */
    unsigned char    size;           /* number of uchar/ushort/ulong */
    unsigned char    buffer[TIP630_MAXIOBUF]; /* buffer where data is stored */
} 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"

int fd;
int result;
TIP630_IO_BUF ioBuf;
unsigned long *pulPtr;

...

/*
** Read buffer from ID-Space. 10 longwords from offset 0x10..0x19
*/
ioBuf.space = TIP630_IDSPACE;
ioBuf.offset = 0x10;
ioBuf.size = 10;

result = devctl( fd,
                 DCMD_TIP630_READ_ULONG,
                 &ioBuf,
                 sizeof(TIP630_IO_BUF),
                 NULL);

if (result == EOK)
{
    pulPtr = (unsigned long*)ioBuf.buffer;
    for (i = 0; i < ioBuf.size; i++) {
        printf("0x%08lx\n", pulPtr[i]);
    }
}
```

## ERRORS

EINVAL  
ENOMEM

The specified parameters are invalid.  
The supplied buffer is too small.

## SEE ALSO

Library Reference - devctl()

### 3.3.4 DCMD\_TIP630\_WRITE\_UCHAR

#### NAME

DCMD\_TIP630\_WRITE\_UCHAR - Write a number of bytes (8-bit) to a specified address

#### DESCRIPTION

This devctl function writes a number of bytes (8-bit) to a specified address. A pointer to the caller's buffer (*TIP630\_IO\_BUF*) and the size of this structure is passed by the parameters *data\_ptr* and *n\_bytes* to the device driver.

The *TIP630\_IO\_BUF* structure has the following layout:

```
typedef struct
{
    unsigned char    space;           /* address space to read from */
    unsigned char    offset;         /* address offset in space */
    unsigned char    size;           /* number of uchar/ushort/ulong */
    unsigned char    buffer[TIP630_MAXIOBUF]; /* buffer where data is stored */
} 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 written

#### *buffer*

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

## EXAMPLE

```
#include "tip630.h"

int fd;
int result;
TIP630_IO_BUF    ioBuf;
unsigned char     *pucPtr;

...

/*
** Write buffer to IO-Space. 4 bytes from offset 0x10..0x13
*/
ioBuf.space      = TIP630_IOSPACE;
ioBuf.offset     = 0x10;
ioBuf.size      = 4;
pucPtr          = (unsigned char*)ioBuf.buffer;
pucPtr[0]       = 0x01;
pucPtr[1]       = 0x02;
pucPtr[2]       = 0x03;
pucPtr[3]       = 0x04;

result = devctl(   fd,
                  DCMD_TIP630_WRITE_UCHAR,
                  &ioBuf,
                  sizeof(TIP630_IO_BUF),
                  NULL);

if (result != EOK)
{
    /* handle devctl error */
}
```

## ERRORS

EINVAL  
ENOMEM

The specified parameters are invalid.  
The supplied buffer is too small.

## SEE ALSO

Library Reference - devctl()

### 3.3.5 DCMD\_TIP630\_WRITE\_USHORT

#### NAME

DCMD\_TIP630\_WRITE\_USHORT - Write a number of words (16-bit) to a specified address

#### DESCRIPTION

This devctl function writes a number of words (16-bit) to a specified address. A pointer to the caller's buffer (*TIP630\_IO\_BUF*) and the size of this structure is passed by the parameters *data\_ptr* and *n\_bytes* to the device driver.

The *TIP630\_IO\_BUF* structure has the following layout:

```
typedef struct
{
    unsigned char    space;           /* address space to read from */
    unsigned char    offset;         /* address offset in space */
    unsigned char    size;           /* number of uchar/ushort/ulong */
    unsigned char    buffer[TIP630_MAXIOBUF]; /* buffer where data is stored */
} 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 written

#### *buffer*

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

## EXAMPLE

```
#include "tip630.h"

int fd;
int result;
TIP630_IO_BUF      ioBuf;
unsigned short     *pusPtr;

...

/*
** Write buffer to IO-Space. 4 words from offset 0x10..0x17
*/
ioBuf.space      = TIP630_IOSPACE;
ioBuf.offset     = 0x10;
ioBuf.size       = 4;
pusPtr          = (unsigned short*)ioBuf.buffer;
pusPtr[0]        = 0x0102;
pusPtr[1]        = 0x0304;
pusPtr[2]        = 0x0506;
pusPtr[3]        = 0x0708;

result = devctl(   fd,
                  DCMD_TIP630_WRITE_USHORT,
                  &ioBuf,
                  sizeof(TIP630_IO_BUF),
                  NULL);

if (result != EOK)
{
    /* handle devctl error */
}
```

## ERRORS

EINVAL  
ENOMEM

The specified parameters are invalid.  
The supplied buffer is too small.

## SEE ALSO

Library Reference - devctl()

### 3.3.6 DCMD\_TIP630\_WRITE\_ULONG

#### NAME

DCMD\_TIP630\_WRITE\_ULONG - Write a number of longwords (32-bit) to a specified address

#### DESCRIPTION

This devctl function writes a number of longwords (32-bit) to a specified address. A pointer to the caller's buffer (*TIP630\_IO\_BUF*) and the size of this structure is passed by the parameters *data\_ptr* and *n\_bytes* to the device driver.

The *TIP630\_IO\_BUF* structure has the following layout:

```
typedef struct
{
    unsigned char    space;           /* address space to read from */
    unsigned char    offset;         /* address offset in space */
    unsigned char    size;           /* number of uchar/ushort/ulong */
    unsigned char    buffer[TIP630_MAXIOBUF]; /* buffer where data is stored */
} 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 written

#### *buffer*

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

## EXAMPLE

```
#include "tip630.h"

int fd;
int result;
TIP630_IO_BUF ioBuf;
unsigned long *pulVal;

...

/*
** Write buffer to IO-Space. 4 longwords from offset 0x10..0x1F
*/
ioBuf.space = TIP630_IOSPACE;
ioBuf.offset = 0x10;
ioBuf.size = 4;
pulPtr = (PULONG)RWBuf.buffer;
pulPtr[0] = 0x01020304;
pulPtr[1] = 0x05060708;
pulPtr[2] = 0x090a0b0c;
pulPtr[3] = 0x0d0e0f00;

result = devctl( fd,
                 DCMD_TIP630_WRITE_ULONG,
                 &ioBuf,
                 sizeof(TIP630_IO_BUF),
                 NULL);

if (result != EOK)
{
    /* handle devctl error */
}
```

## ERRORS

EINVAL  
ENOMEM

The specified parameters are invalid.  
The supplied buffer is too small.

## SEE ALSO

Library Reference - devctl()

### 3.3.7 DCMD\_TIP630\_RECONFIGURE

#### NAME

DCMD\_TIP630\_RECONFIGURE – Reconfigure IP-clock and bus-width

#### DESCRIPTION

This *devctl* function reconfigures the setup of the IP slot where the TIP630 resides. The IP-clock and the bus-width can be changed. A pointer to the caller's buffer (*TIP630\_RECONFIG\_BUF*) and the size of this structure is passed by the parameters *data\_ptr* and *n\_bytes* to the device driver.

The *TIP630\_RECONFIG\_BUF* structure has the following layout:

```
typedef struct
{
    int          busWidth;    /* bus width (8 or 16 bit)    */
    int          ipClock;    /* IP clock (8 or 32 MHz)    */
} TIP630_RECONFIG_BUF, *PTIP630_RECONFIG_BUF;
```

#### *busWidth*

This value specifies the width of the IP-Bus.

TIP630_BUSWIDTH_8BIT	8-bit Bus
TIP630_BUSWIDTH_16BIT	16-bit Bus (default)

#### *ipClock*

This value specifies the IP-Clock speed.

TIP630_IPCLOCK_8MHZ	8 MHz (default)
TIP630_IPCLOCK_32MHZ	32 MHz

## EXAMPLE

```
int fd;
int result;
TIP630_RECONFIG_BUF configBuf;

/*
** Configure IPAC slot for 16-Bit Bus and 8MHz IP clock
*/
configBuf.busWidth      = TIP630_BUSWIDTH_16BIT;
configBuf.ipClock       = TIP630_IPCLOCK_8MHZ;

result = devctl(   fd,
                  DCMD_TIP630_RECONFIGURE,
                  &configBuf,
                  sizeof(TIP630_RECONFIG_BUF),
                  NULL);

if (result != EOK)
{
    /* handle devctl error */
}

...
```

## ERRORS

EINVAL

This error code is returned if the size of the message buffer is too small.

ETIMEDOUT

The allowed time to finish the write request elapsed.

ENETDOWN

The CAN-controller is in bus of state.

EBUSY

There is an other request send a message on this device.

## SEE ALSO

Library Reference - devctl()

### 3.3.8 DCMD\_TIP630\_SET\_CLOCK

#### NAME

DCMD\_TIP630\_SET\_CLOCK – Set values for clock setup

#### DESCRIPTION

This devctl function configures both local clocks of the TIP630. The necessary data should be taken from a tool called 'Cypress Cyberclocks', which calculates the correct register settings for the onboard Cypress clock device. A pointer to the caller's parameter buffer (*TIP630\_CLOCK\_PARAM*) is passed by the argument *data\_ptr* to the device driver.

The *TIP630\_CLOCK\_PARAM* structure has the following layout:

```
typedef struct _TIP630_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          PCount;         // 1bit
    UCHAR          Qcount;         // 7bit
    UCHAR          ClkASrc;        // 3bit
    UCHAR          ClkBsrc;        // 3bit
    UCHAR          Div2Src;        // 1bit
    UCHAR          Div2N;         // 7bit
} TIP630_CLOCK_PARAM, *PTIP630_CLOCK_PARAM;
```

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

## EXAMPLE

```
int fd;
int result;
TIP630_CLOCK_PARAM clockBuf =
    {0x03, 0x00, 0x32, 0x01, 0x00, 0x01, 0x0008, 0x01,
     0x06, 0x01, 0x04, 0x00, 0x19};

...

/*
** Configure clocks (2MHz/4MHz)
*/
result = devctl(    fd,
                   DCMD_TIP630_SET_CLOCK,
                   &clockBuf,
                   sizeof(TIP630_CLOCK_PARAM),
                   NULL);

if (result != EOK)
{
    /* handle devctl error */
}

...
```

## ERRORS

EINVAL  
EACCES

The specified buffer is invalid.  
The programming of the clock device failed.

## SEE ALSO

Library Reference - devctl()

### 3.3.9 DCMD\_TIP630\_PROGRAM\_FPGA

#### NAME

DCMD\_TIP630\_PROGRAM\_FPGA – Program the contents of the FPGA

#### DESCRIPTION

This *devctl* function programs the supplied data into the FPGA's logic cells. The buffer must be an array of unsigned char values that contains the FPGA data, which is supplied to the device driver via a shared memory object. The length of the buffer is fixed to *SIZE\_OF\_FPGA\_DATA* (0x28C44 bytes). A pointer to the caller's parameter buffer (*TIP630\_FPGAPROG\_BUF*) is passed by the argument *data\_ptr* to the device driver.

The *TIP630\_FPGAPROG\_BUF* structure has the following layout:

```
typedef struct
{
    char                shMemName[TIP630_MAXNAMELEN];
} TIP630_FPGAPROG_BUF, *PTIP630_FPGAPROG_BUF;
```

#### *shMemName*

This value specifies the name of the shared memory object which is used to transfer the FPGA data buffer to the device driver.

**The shared memory object must be created and initialized by the user application.**

#### EXAMPLE

```
#include <sys/mman.h>
#include "tip630.h"

int fd;
int result;
unsigned char    mybytereord[] = ...;    /* FPGA data */
int              sharedmemfd;
TIP630_FPGAPROG_BUF    FpgaBuf;

/*
** init shared-memory buffer
*/
sharedmemfd = shm_open("/fpgabuffer", O_RDWR | O_CREAT, 0777);
if (sharedmemfd == -1)
{
```

```
    printf("Open of shared memory failed: %s\n", strerror(errno));
}

/* set size of shared memory section */
if (ftruncate(sharedmemfd, SIZE_OF_FPGA_DATA) == -1)
{
    printf("error ftruncate: %s\n", strerror(errno));
}

/* map memory area */
pucPtr = mmap( 0,
              SIZE_OF_FPGA_DATA, PROT_READ | PROT_WRITE,
              MAP_SHARED,
              sharedmemfd,
              0);
if (pucPtr == MAP_FAILED)
{
    printf("mmap failed: %s\n", strerror(errno));
}

/*
** copy the whole buffer into shared memory
*/
memcpy( pucPtr, &mybyterecord[0], SIZE_OF_FPGA_DATA );

/*
** start programming of FPGA
*/
sprintf( FpgaBuf.shMemName, "/fpgabuffer" );
result = devctl( fd,
                DCMD_TIP630_PROGRAM_FPGA,
                &FpgaBuf,
                sizeof(TIP630_FPGAPROG_BUF),
                NULL);
if (result != EOK)
{
    /* handle devctl error */
}
}
```

## ERRORS

EINVAL  
EACCES

The specified buffer is invalid.  
The programming of the FPGA failed.

## SEE ALSO

Library Reference - devctl()

### 3.3.10 DCMD\_TIP630\_MEMSPACE\_MAP

#### NAME

DCMD\_TIP630\_MEMSPACE\_MAP – Map the TIP630’s memory space

#### DESCRIPTION

This *devctl* function maps available memory space of the TIP630. The amount of memory must be supplied to this function. A pointer to the caller’s parameter buffer (*unsigned long*) is passed by the argument *data\_ptr* to the device driver.

#### EXAMPLE

```
int fd;
int result;
unsigned long memSize;

...

/*
** map 0x100 bytes of memory space
*/
memSize = 0x100;
result = devctl(    fd,
                  DCMD_TIP630_MEMSPACE_MAP,
                  &memSize,
                  sizeof(unsigned long),
                  NULL);

if (result != EOK)
{
    /* handle devctl error */
}

...
```

#### ERRORS

EINVAL	The specified buffer is invalid.
EBUSY	Memory is already mapped.
EACCES	The mapping has failed.

#### SEE ALSO

Library Reference - *devctl()*

### 3.3.11 DCMD\_TIP630\_MEMSPACE\_UNMAP

#### NAME

DCMD\_TIP630\_MEMSPACE\_UNMAP – Unmap the TIP630's memory space

#### DESCRIPTION

This *devctl* function unmaps previously mapped memory space of the TIP630. No parameter is necessary for this function. The arguments *data\_ptr* and *n\_bytes* are unused and should be set to *NULL* respective 0.

#### EXAMPLE

```
int fd;
int result;

...

/*
** unmap the previously mapped memory space
*/
result = devctl(    fd,
                   DCMD_TIP630_MEMSPACE_UNMAP,
                   NULL,
                   0,
                   NULL);

if (result != EOK)
{
    /* handle devctl error */
}

...
```

#### ERRORS

EACCES	There is no memory mapped.
--------	----------------------------

#### SEE ALSO

Library Reference - *devctl()*