
TIP120-SW-42

VxWorks Device Driver

Motion Controller with Incremental Encoder Interface

Version 2.0.x

User Manual

Issue 2.0.0

November 2008

TEWS TECHNOLOGIES GmbH

Am Bahnhof 7
25469 Halstenbek, Germany
www.tews.com

Phone: +49 (0) 4101 4058 0
Fax: +49 (0) 4101 4058 19
e-mail: info@tews.com

TEWS TECHNOLOGIES LLC

9190 Double Diamond Parkway,
Suite 127, Reno, NV 89521, USA
www.tews.com

Phone: +1 (775) 850 5830
Fax: +1 (775) 201 0347
e-mail: usasales@tews.com

TIP120-SW-42

VxWorks Device Driver

Motion Controller with Incremental Encoder Interface

Supported Modules:
TIP120

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.

©1997-2008 by TEWS TECHNOLOGIES GmbH

Issue	Description	Date
1.0	First Issue	November 1997
1.1	General Revision	September 2003
2.0.0	TEWS Carrier Support added, application interface modified	November 28, 2008

Table of Contents

1	INTRODUCTION.....	4
1.1	Device Driver	4
1.2	IPAC Carrier Driver	5
2	INSTALLATION.....	6
2.1	Include device driver in VxWorks projects	6
2.2	System resource requirement	7
3	I/O SYSTEM FUNCTIONS.....	8
3.1	tip120Drv()	8
3.2	tip120DevCreate().....	10
4	I/O FUNCTIONS	13
4.1	open()	13
4.2	close().....	15
4.3	ioctl()	17
4.3.1	FIO_TIP120_CMD_WRITE.....	19
4.3.2	FIO_TIP120_CMD_READ	21
4.3.3	FIO_TIP120_STAT_READ	23
4.3.4	FIO_TIP120_ENABLE_INT	25
4.3.5	FIO_TIP120_DISABLE_INT	28
4.3.6	FIO_TIP120_READ_INPUT.....	30
4.3.7	FIO_TIP120_SET_OUTPUT.....	32

1 Introduction

1.1 Device Driver

The TIP120-SW-42 VxWorks device driver software allows the operation of the supported IPAC modules conforming to the VxWorks I/O system specification. This includes a device-independent basic I/O interface with *open()*, *close()* and *ioctl()* functions.

The TIP120-SW-42 device driver supports the following features:

- Execute read and write commands on the LM628/LM629 motion controller
- Setting output lines
- Reading state of input lines
- Enable and configure interrupt callback functions (input lines and LM628/LM629 interrupts)

The TIP120-SW-42 supports the modules listed below:

TIP120-x0	2 axes motion controller (LM628/LM629)	IndustryPack
TIP120-x1	1 axes motion controller (LM628/LM629)	IndustryPack

In this document all supported modules and devices will be called TIP120. Specials for certain devices will be advised.

To get more information about the features and use of supported devices it is recommended to read the manuals listed below.

- TIP120 User Manual
- TIP120 Engineering Manual
- CARRIER-SW-42 IPAC Carrier User Manual
- LM628/LM629 motion controller Manuals

1.2 IPAC Carrier Driver

IndustryPack (IPAC) carrier boards have different implementations of the system to IndustryPack bus bridge logic, different implementations of interrupt and error handling and so on. Also the different byte ordering (big-endian versus little-endian) of CPU boards will cause problems on accessing the IndustryPack I/O and memory spaces.

To simplify the implementation of IPAC device drivers which work with any supported carrier board, TEWS TECHNOLOGIES has designed a so called Carrier Driver that hides all differences of different carrier boards under a well defined interface.

The TEWS TECHNOLOGIES IPAC Carrier Driver CARRIER-SW-42 is part of this TIP120-SW-42 distribution. It is located in directory CARRIER-SW-42 on the corresponding distribution media.

This IPAC Device Driver requires a properly installed IPAC Carrier Driver. Due to the design of the Carrier Driver, it is sufficient to install the IPAC Carrier Driver once, even if multiple IPAC Device Drivers are used.

Please refer to the CARRIER-SW-42 User Manual for a detailed description how to install and setup the CARRIER-SW-42 device driver, and for a description of the TEWS TECHNOLOGIES IPAC Carrier Driver concept.

How to use the carrier driver in the application program is shown in the programming example tip120exa.c.

2 Installation

Following files are located on the distribution media:

Directory path 'TIP120-SW-42':

tip120drv.c	TIP120 device driver source
tip120def.h	TIP120 driver include file
tip120.h	TIP120 include file for driver and application
tip120exa.c	Example application
ipac_carrier.h	Carrier driver interface definitions
TIP120-SW-42-2.0.0.pdf	PDF copy of this manual
ChangeLog.txt	Release history
Release.txt	Release information

2.1 Include device driver in VxWorks projects

For including the TIP120-SW-42 device driver into a VxWorks project (e.g. Tornado IDE or Workbench) follow the steps below:

- (1) Copy the files from the distribution media into a subdirectory in your project path.
(For example: ./TIP120)
- (2) Add the device drivers C-files to your project.
- (3) Now the driver is included in the project and will be built with the project.

For a more detailed description of the project facility please refer to your VxWorks User's Guide (e.g. Tornado, Workbench, etc.)

2.2 System resource requirement

The table gives an overview over the system resources that will be needed by the driver.

Resource	Driver requirement	Devices requirement
Memory	< 1 KB	< 1 KB
Stack	< 1 KB	---

Memory and Stack usage may differ from system to system, depending on the used compiler and its setup.

The following formula shows the way to calculate the common requirements of the driver and devices.

$$\langle total\ requirement \rangle = \langle driver\ requirement \rangle + (\langle number\ of\ devices \rangle * \langle device\ requirement \rangle)$$

The maximum usage of some resources is limited by adjustable parameters. If the application and driver exceed these limits, increase the according values in your project.

3 I/O system functions

This chapter describes the driver-level interface to the I/O system. The purpose of these functions is to install the driver in the I/O system, add and initialize devices.

3.1 tip120Drv()

NAME

tip120Drv() - installs the TIP120 driver in the I/O system

SYNOPSIS

```
#include "tip120.h"
```

```
STATUS tip120Drv(void)
```

DESCRIPTION

This function installs the TIP120 driver in the I/O system.

A call to this function is the first thing the user has to do before adding any device to the system or performing any I/O request.

EXAMPLE

```
#include "tip120.h"

STATUS          result;

/*-----
   Initialize Driver
   -----*/
result = tip120Drv();
if (result == ERROR)
{
    /* Error handling */
}
```

RETURNS

OK or ERROR. If the function fails an error code will be stored in *errno*.

ERROR CODES

Error codes are only set by system functions. The error codes are stored in *errno* and can be read with the function *errnoGet()*.

SEE ALSO

VxWorks Programmer's Guide: I/O System

3.2 tip120DevCreate()

NAME

tip120DevCreate() – Add a TIP120 device to the VxWorks system

SYNOPSIS

```
#include "tip120.h"
```

```
STATUS tip120DevCreate  
(  
    char      *name,  
    int       devIdx,  
    int       funcType,  
    void      *pParam  
)
```

DESCRIPTION

This routine adds the selected device to the VxWorks system. The device hardware will be setup and prepared for use.

This function must be called before performing any I/O request to this device.

PARAMETER

name

This string specifies the name of the device that will be used to identify the device, for example for *open()* calls.

devIdx

This index number (zero based) specifies the TIP120 minor device number to add to the system.

If modules of the same type are installed the device numbers will be assigned in the order the IPAC CARRIER *ipFindDevice()* function will find the devices.

funcType

This parameter is unused and should be set to 0.

pParam

This parameter points to a structure (*TIP120_DEVCONFIG*) containing the configuration of the device.

```
typedef struct
{
    struct ipac_resource    *ipac;
} TIP120_DEVCONFIG;
```

ipac

Pointer to TIP120 module resource descriptor, retrieved by CARRIER Driver ipFindDevice() function

EXAMPLE

```
#include "tip120.h"

STATUS                result;
TIP120_DEVCONFIG     tip120Conf;
struct ipac_resource  ipac;

/* IPAC CARRIER Driver initialization */

/*
** Find an IP module from TEWS TECHNOLOGIES (manufacturer = 0xB3)
** with model number MODEL_TIP120 (see tip120.h).
** This module uses both interrupt lines and needs an IACK cycle,
** we need only the IO space base address for the related driver.
*/
result = ipFindDevice( 0xB3, MODEL_TIP120, 0,
                      IPAC_INT0_EN | IPAC_INT1_EN | IPAC_LEVEL_SENS |
                      IPAC_IACK_CYC | IPAC_CLK_8MHZ, &ipac);

if (result == OK)
{
    devConfig.ipac = &ipac;

    /*-----
    Create the device "/tip120/0" for the first device
    -----*/
    tip120Conf.ipac = &ipac;

    ...
}
```

```
...

    result = tip102DevCreate(    "/tip120/0",
                                0,
                                0,
                                (void*)&tip120Conf);

    if (result == OK)
    {
        /* Device successfully created */
    }
    else
    {
        /* Error occurred when creating the device */
    }
}
else
{
    /* No IP found on supported IP carrier boards */
}
}
```

RETURNS

OK or ERROR. If the function fails an error code will be stored in *errno*.

ERROR CODES

The error codes are stored in *errno* and can be read with the function *errnoGet()*.

Error code	Description
S_ioLib_NO_DRIVER	The driver has not been started
EINVAL	Invalid input argument
EISCONN	The device has already been created
ENOTSUP	IPAC with unsupported model version specified
ETIMEDOUT	Initial LM628/LM629 commands timed out (probably HW-error)

SEE ALSO

VxWorks Programmer's Guide: I/O System

4 I/O Functions

4.1 open()

NAME

open() - open a device or file.

SYNOPSIS

```
int open
(
    const char *name,
    int        flags,
    int        mode
)
```

DESCRIPTION

Before I/O can be performed to the TIP120 device, a file descriptor must be opened by invoking the basic I/O function *open()*.

PARAMETER

name

Specifies the device which shall be opened, the name specified in tip120DevCreate() must be used

flags

Not used

mode

Not used

EXAMPLE

```
int      fd;

/*-----
   Open the device named "/tip120/0" for I/O
   -----*/
fd = open("/tip120/0", 0, 0);
if (fd == ERROR)
{
    /* Handle error */
}
```

RETURNS

A device descriptor number or ERROR. If the function fails an error code will be stored in *errno*.

ERROR CODES

The error code can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual)

SEE ALSO

ioLib, basic I/O routine - *open()*

4.2 close()

NAME

close() – close a device or file

SYNOPSIS

```
STATUS close
(
    int      fd
)
```

DESCRIPTION

This function closes opened devices.

PARAMETER

fd

This file descriptor specifies the device to be closed. The file descriptor has been returned by the *open()* function.

EXAMPLE

```
int      fd;
STATUS   retval;

/*-----
   close the device
   -----*/
retval = close(fd);
if (retval == ERROR)
{
    /* Handle error */
}
```

RETURNS

OK or ERROR. If the function fails, an error code will be stored in *errno*.

ERROR CODES

The error code can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual).

SEE ALSO

ioLib, basic I/O routine - close()

4.3 ioctl()

NAME

ioctl() - performs an I/O control function.

SYNOPSIS

```
#include "tip120.h"
```

```
int ioctl
(
    int    fd,
    int    request,
    int    arg
)
```

DESCRIPTION

Special I/O operation that do not fit to the standard basic I/O calls (read, write) will be performed by calling the ioctl() function.

PARAMETER

fd

This file descriptor specifies the device to be used. The file descriptor has been returned by the *open()* function.

request

This argument specifies the function that shall be executed. Following functions are defined:

Function	Description
FIO_TIP120_CMD_WRITE	execute a write command on the LM628/LM629 motion controller
FIO_TIP120_CMD_READ	execute a read command on the LM628/LM629 motion controller and return value
FIO_TIP120_STAT_READ	read state of the LM628/LM629 motion controller
FIO_TIP120_ENABLE_INT	Enable and configure interrupt callback for input line or LM628/LM629 motion controller interrupt
FIO_TIP120_DISABLE_INT	Disable callback for an interrupt
FIO_TIP120_READ_INPUT	Read state of the input lines
FIO_TIP120_SET_OUTPUT	Set output line

arg

This parameter depends on the selected function (request). How to use this parameter is described below with the function.

RETURNS

OK or ERROR. If the function fails an error code will be stored in *errno*.

ERROR CODES

The error code can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual).

SEE ALSO

ioLib, basic I/O routine - *ioctl()*

4.3.1 FIO_TIP120_CMD_WRITE

This I/O control function executes a write command on a motion controller where a specified number of data bytes will be written. The function specific control parameter **arg** is a pointer on a *TIP120_CMD_BUFFER*.

```
typedef struct
{
    unsigned char    channel;
    unsigned char    command;
    unsigned char    dataLen;
    unsigned char    data[14];
} TIP120_CMD_BUFFER;
```

channel

This parameter specifies the local channel number (axis) on the TIP120. Allowed channel numbers are 1 and 2 for TIP120-x0 and only 1 for TIP120-x1.

command

This parameter specifies the command that shall be executed on the LM628/LM629 motion controller. For valid commands refer to the LM628/LM629 motion controller manuals. (For more transparent applications definitions for valid command codes are predefined in tip120.h)

dataLen

This parameter specifies the number of valid data bytes in *data[]*. These data bytes will be written to the LM628/LM629 motion controller. The number of data bytes depends on the specified command. Please refer to the LM628/LM629 motion controller manuals for the right data length.

data[]

This array contains the data that shall be written. The array length is 14 bytes. This is THE maximum data length a valid command may need.

EXAMPLE

```
#include "tip120.h"

int                fd;
TIP120_CMD_BUFFER cmdBuf;
int                retval;

/*-----
   Write a MSKI (Mask Interrupt (1Ch)) command to 1st axis
   - Enable trajectory complete interrupt
   -----*/
cmdBuf.channel = 1;
cmdBuf.command = 0x1C;
cmdBuf.dataLen = 2;
cmdBuf.data[0] = 0x00; /* unused */
cmdBuf.data[1] = 0x04; /* enable Trajectory complete interrupt */

retval = ioctl(fd, FIO_TIP120_CMD_WRITE, (int)&cmdBuf);
if (retval != ERROR)
{
    /* function succeeded */
}
else
{
    /* handle the error */
}
```

ERROR CODES

EINVAL	Invalid or missing input parameter (e.g. invalid channel number, or argument pointer is NULL)
ETIMEDOUT	The command sequence timed out

4.3.2 FIO_TIP120_CMD_READ

This I/O control function executes a read command on a motion controller and reads the specified number of data bytes. The function specific control parameter **arg** is a pointer on a *TIP120_CMD_BUFFER*.

```
typedef struct
{
    unsigned char    channel;
    unsigned char    command;
    unsigned char    dataLen;
    unsigned char    data[14];
} TIP120_CMD_BUFFER;
```

channel

This parameter specifies the local channel number (axis) on the TIP120. Allowed channel numbers are 1 and 2 for TIP120-x0 and only 1 for TIP120-x1.

command

This parameter specifies the command that shall be executed on the LM628/LM629 motion controller. For valid commands refer to the LM628/LM629 motion controller manuals. (For more transparent applications definitions for valid command codes are predefined in tip120.h)

dataLen

This parameter specifies the number of data bytes that shall be returned in *data[]*. These data bytes will be read from the LM628/LM629 motion controller. The number of returned data bytes depends on the specified command. Please refer to the LM628/LM629 motion controller manuals for the right data length.

data[]

This array contains the data that shall be written. The array length is 14 bytes. This is the maximum data length a valid command may need.

EXAMPLE

```
#include "tip120.h"

int                fd;
TIP120_CMD_BUFFER cmdBuf;
int               retval;
unsigned int      position;

/*-----
   Execute a RDRP (Read real position (0Ah)) command on 1st axis
   -----*/
cmdBuf.channel = 1;
cmdBuf.command = 0x0A;
cmdBuf.dataLen = 4;

retval = ioctl(fd, FIO_TIP120_CMD_READ, (int)&cmdBuf);
if (retval != ERROR)
{
    /* function succeeded */
    position = (((unsigned int)cmdBuf.data[0]) << 24) & 0xFF000000 |
               (((unsigned int)cmdBuf.data[1]) << 16) & 0xFF0000 |
               (((unsigned int)cmdBuf.data[2]) << 8) & 0xFF00 |
               (((unsigned int)cmdBuf.data[3]) << 0) & 0xFF);
}
else
{
    /* handle the error */
}
```

ERROR CODES

EINVAL	Invalid or missing input parameter (e.g. invalid channel number, or argument pointer is NULL)
ETIMEDOUT	The command sequence timed out

4.3.3 FIO_TIP120_STAT_READ

This I/O control function reads the state of the motion controller. The function specific control parameter **arg** is a pointer on a *TIP120_IO_BUFFER*.

```
typedef struct
{
    unsigned char    channel;
    unsigned char    value;
} TIP120_IO_BUFFER;
```

channel

This parameter specifies the local channel number (axis) on the TIP120. Allowed channel numbers are 1 and 2 for TIP120-x0 and only 1 for TIP120-x1.

value

This parameter returns the state of the LM628/LM629 motion controller. For meaning of the state flags refer to the LM628/LM629 motion controller manuals. The following flags are defined:

Flag	Description
TIP120_S_BUSY	Busy Bit
TIP120_S_CMD_ERR	Command Error [Interrupt]
TIP120_S_TRJ_CMP	Trajectory Complete [Interrupt]
TIP120_S_INDEX	Index Pulse Observed [Interrupt]
TIP120_S_WRAP	Wraparound Occurred [Interrupt]
TIP120_S_POS_ERR	Excessive Position Error [Interrupt]
TIP120_S_BRKPNT	Breakpoint Reached [Interrupt]
TIP120_S_MOTOR	Motor Off

EXAMPLE

```
#include "tip120.h"

int                fd;
TIP120_IO_BUFFER  statBuf;
int               retval;

/*-----
   read the state of the motion controller on 2nd channel
   -----*/
statBuf.channel = 2;

retval = ioctl(fd, FIO_TIP120_STAT_READ, (int)&statBuf);
if (retval != ERROR)
{
    /* function succeeded */
    printf("STAT: %Xh\n", statBuf.value);
}
else
{
    /* handle the error */
}
```

ERROR CODES

EINVAL	Invalid or missing input parameter (e.g. invalid channel number, or argument pointer is NULL)
--------	---

4.3.4 FIO_TIP120_ENABLE_INT

This I/O control function configures interrupt handling and enables the specified interrupt for the axis. If an enabled interrupt event occurs, the driver will call an application defined callback interrupt function with an application defined parameter. Every interrupt source can call a different function.

Interrupt events generated on the LM628/LM629 motion controller have to be enabled separately with the write command (MSKI)

The function specific control parameter **arg** is a pointer on a *TIP120_INT_BUFFER*.

```
typedef struct
{
    unsigned char    channel;
    int              intIdx;
    int              intTrans;
    void             *intCallback;
    int              intParam;
} TIP120_INT_BUFFER;
```

channel

This parameter specifies the local channel number (axis) on the TIP120. Allowed channel numbers are 1 and 2 for TIP120-x0 and only 1 for TIP120-x1.

intIdx

This value specifies the interrupt that shall be enabled. The following table shows the index of the interrupt events (defined in tip120.h):

interrupt index	description
TIP120_I_CMD_ERR	Command-Error Interrupt (from LM628/LM629)
TIP120_I_TRJ_CMP	Trajectory-Complete Interrupt (from LM628/LM629)
TIP120_I_INDEX	Index-Pulse Interrupt (from LM628/LM629)
TIP120_I_WRAP	Wrap-Around Interrupt (from LM628/LM629)
TIP120_I_POS_ERR	Position-Error Interrupt (from LM628/LM629)
TIP120_I_BRKPNT	Breakpoint Interrupt (from LM628/LM629)
TIP120_I_INPUT1	Interrupt event on INPUT1
TIP120_I_INPUT2	Interrupt event on INPUT2
TIP120_I_INPUT3	Interrupt event on INPUT3

intTrans

This parameter specifies the transition the input event shall be detected on, it is unused for interrupts coming from the LM628/LM629. Allowed values are *TIP120_T_HIGH* for a low-to-high transition and *TIP120_T_LOW* for a high-to-low transition. The values are defined in tip120.h.

intCallback

This parameter shall point to the callback function that will be called if the specified event occurs. The callback-function shall have the following interface:

```
void cbfName(int param);
```

The function will be called in interrupt context. Blocking functions are not allowed, loops shall be avoided and the execution time shall be kept short.

intParam

This parameter value will be passed to the application callback function specified with *intCallback*.

EXAMPLE

```
#include "tip120.h"

int                fd;
TIP120_INT_BUFFER intBuf;
int                retval;

/*-----
   Enable Interrupt on INPUT1 high transition of the 1st axis
   -----*/
intBuf.channel = 1;
intBuf.intIdx = TIP120_I_INPUT1;
intBuf.intTrans = TIP120_T_HIGH;
intBuf.intCallback = in1CBFunc;
intBuf.intParam = 0x1234;

retval = ioctl(fd, FIO_TIP120_ENABLE_INT, (int)&intBuf);
if (retval != ERROR)
{
    /* function succeeded */
}
else
{
    /* handle the error */
}

...
```

```
...

/* Interrupt callback function for INPUT1 */
void inlCBFunc
(
    int      intParam
)
{
    ...
    return;
}
```

ERROR CODES

EINVAL	Invalid or missing input parameter (e.g. invalid channel number, or argument pointer is NULL)
EBUSY	The specified interrupt has already been enabled, disable the interrupt before enabling again

4.3.5 FIO_TIP120_DISABLE_INT

This I/O control function disables the specified interrupt for the axis and removes the callback function from the specified interrupt.

Interrupt events generated on the LM628/LM629 motion controller have to be disabled separately with the write command (MSKI)

The function specific control parameter **arg** is a pointer on a *TIP120_INT_BUFFER*.

typedef struct

```
{
    unsigned char    channel;
    int              intIdx;
    int              intTrans;
    void             *intCallback;
    int              intParam;
} TIP120_INT_BUFFER;
```

channel

This parameter specifies the local channel number (axis) on the TIP120. Allowed channel numbers are 1 and 2 for TIP120-x0 and only 1 for TIP120-x1.

intIdx

This value specifies the interrupt that shall be disabled. The following table shows the index of the interrupt events (defined in tip120.h):

interrupt index	description
TIP120_I_CMD_ERR	Command-Error Interrupt (from LM628/LM629)
TIP120_I_TRJ_CMP	Trajectory-Complete Interrupt (from LM628/LM629)
TIP120_I_INDEX	Index-Pulse Interrupt (from LM628/LM629)
TIP120_I_WRAP	Wrap-Around Interrupt (from LM628/LM629)
TIP120_I_POS_ERR	Position-Error Interrupt (from LM628/LM629)
TIP120_I_BRKPNT	Breakpoint Interrupt (from LM628/LM629)
TIP120_I_INPUT1	Interrupt event on INPUT1
TIP120_I_INPUT2	Interrupt event on INPUT2
TIP120_I_INPUT3	Interrupt event on INPUT3

intTrans

This parameter will not be used.

intCallback

This parameter will not be used.

intParam

This parameter will not be used.

EXAMPLE

```
#include "tip120.h"

int                fd;
TIP120_INT_BUFFER intBuf;
int                retval;

/*-----
   Disable Interrupt on INPUT1 high transition of the 1st axis
   -----*/
intBuf.channel = 1;
intBuf.intIdx = TIP120_I_INPUT1;

retval = ioctl(fd, FIO_TIP120_DISABLE_INT, (int)&intBuf);
if (retval != ERROR)
{
    /* function succeeded */
}
else
{
    /* handle the error */
}
```

ERROR CODES

EINVAL	Invalid or missing input parameter (e.g. invalid channel number, or argument pointer is NULL)
--------	---

4.3.6 FIO_TIP120_READ_INPUT

This I/O control function reads the state of the input lines of the specified axis. The function specific control parameter **arg** is a pointer on a *TIP120_IO_BUFFER*.

```
typedef struct
{
    unsigned char      channel;
    unsigned char      value;
} TIP120_IO_BUFFER;
```

channel

This parameter specifies the local channel number (axis) on the TIP120. Allowed channel numbers are 1 and 2 for TIP120-x0 and only 1 for TIP120-x1.

value

This parameter returns the state of input lines. A set bit means an active input. Bit-0 specifies the state of INPUT1, bit-1 specifies the state of INPUT2 and bit-2 specifies the state of INPUT3.

EXAMPLE

```
#include "tip120.h"

int          fd;
TIP120_IO_BUFFER  inBuf;
int          retval;

/*-----
   read the state of the input lines
   -----*/
inBuf.channel = 2;

retval = ioctl(fd, FIO_TIP120_READ_INPUT, (int)&inBuf);
if (retval != ERROR)
{
    /* function succeeded */
    printf("INPUT: %Xh\n", statBuf.value);
}
else
{
    /* handle the error */
}
```

ERROR CODES

EINVAL

Invalid or missing input parameter (e.g. invalid channel number, or argument pointer is NULL)

4.3.7 FIO_TIP120_SET_OUTPUT

This I/O control function sets the output line of the specified axis. The function specific control parameter **arg** is a pointer on a *TIP120_IO_BUFFER*.

```
typedef struct
{
    unsigned char      channel;
    unsigned char      value;
} TIP120_IO_BUFFER;
```

channel

This parameter specifies the local channel number (axis) on the TIP120. Allowed channel numbers are 1 and 2 for TIP120-x0 and only 1 for TIP120-x1.

value

Bit 0 of this parameter specifies the state of the output line. If bit-0 is set, the output is active.

EXAMPLE

```
#include "tip120.h"

int          fd;
TIP120_IO_BUFFER  outBuf;
int          retval;

/*-----
   set output line of channel 1 active
   -----*/
outBuf.channel = 1;
outBuf.value = 1;

retval = ioctl(fd, FIO_TIP120_SET_OUTPUT, (int)&outBuf);
if (retval != ERROR)
{
    /* function succeeded */
}
else
{
    /* handle the error */
}
```

ERROR CODES

EINVAL

Invalid or missing input parameter (e.g. invalid channel number, or argument pointer is NULL)