The Embedded I/O Company

TEWS TECHNOLOGIES

TIP102-SW-82

Linux Device Driver

Motion Controller with Incremental Interface

Version 1.0.x

User Manual

Issue 1.0.0

May 2005

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

TIP102-SW-82

Motion Controller with Incremental Interface

Linux Device Driver

Issue	Description	Date
1.0.0	First Issue	May 11, 2005

Table of Content

1 INTRODUCTION 4
2 INSTALLATION 5
2.1 Build and install the device driver 5
2.2 Uninstall the device driver 6
2.3 Install device driver into the running kernel 6
2.4 Remove device driver from the running kernel 7
2.5 Change Major Device Number 7
3 DEVICE INPUT/OUTPUT FUNCTIONS 8
3.1 open() 8
3.2 close() 10
3.3 ioctl() 11
3.3.1 T102_IOCG_READ_ADC 13
3.3.2 T102_IOCS_WRITE_DAC 15
3.3.3 T102_IOCG_READ_COUNTER 17
3.3.4 T102_IOCS_COUNTER_PRESET 19
3.3.5 T102_IOCG_READ_REGISTER 21
3.3.6 T102_IOCS_WRITE_REGISTER 23
3.3.7 T102_IOCS_SET_REFMODE 25
3.3.8 T102_IOCS_EVENT_WAIT 27

1 Introduction

The TIP102-SW-82 Linux device driver allows the operation of a TIP102 IPAC module on Linux operating systems.

Because the TIP102 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 TIP102 device driver includes the following features:

- Reading actual counter value
- Setting counter preset value
- Setting counter reference mode
- Waiting for counter interrupt event
- Reading and Writing to controller registers
- Reading ADC input value
- Setting DAC output value

2 Installation

The directory TIP102-SW-82 on the distribution media contains the following files:

TIP102-SW-82.pdf	This manual in PDF format
TIP102-SW-82-SRC.tar.gz	GZIP compressed archive with driver source code
Release.txt	Information about the Device Driver Release

The GZIP compressed archive TIP102-SW-82.tar.gz contains the following files and directories:

tip102/tip102.c	Driver source code
tip102/tip102def.h	Driver include file
tip102/tip102.h	Driver include file for application program
tip102/tpmodule.c	Driver independent library
tip102/tpmodule.h	Driver independent library header file
tip102/makenode	Script to create device nodes on the file system
tip102/Makefile	Device driver make file
tip102/example/tip102example.c	Example application
tip102/example/Makefile	Example application make file

In order to perform an installation, extract all files of the archive TIP102-SW-82.tar.gz to the desired target directory.

Before building a new device driver, the TEWS TECHNOLOGIES IPAC carrier driver must be installed properly, because this driver includes the header file *ipac_carrier.h*, which is part of the IPAC carrier driver distribution. Please refer to the IPAC carrier driver user manual in the directory path *CARRIER-SW-82* on the distribution media.

2.1 Build and install the device driver

- Login as *root*
- Change to the target directory
- To create and install the driver in the module directory */lib/modules/<version>/misc* enter:

  # make install

- Also after the first build we have to execute *depmod* to create a new dependency description for loadable kernel modules. This dependency file is later used by *modprobe* to automatically load the correct IPAC carrier driver modules.

  # depmod –aq

2.2 Uninstall the device driver

- Login as *root*
- Change to the target directory
- To remove the driver from the module directory */lib/modules/<version>/misc* enter:

  # make uninstall

- Update kernel module dependency description file

  # depmod –aq

2.3 Install device driver into the running kernel

- To load the device driver into the running kernel, login as root and execute the following commands:

  # modprobe tip102drv

- After the first build or if you are using dynamic major device allocation it's necessary to create new device nodes on the file system. Please execute the script file *makenode* to do this. If your kernel has enabled the new device file system (devfs) then you have to skip running the *makenode* script. Instead of creating device nodes from the script the driver itself takes creating and destroying of device nodes in its responsibility.

  # sh makenode

On success the device driver will create a minor device for each TIP102 module found. The first TIP102 can be accessed with device node /dev/tip102_0, the second TIP102 or the second channel of the first TIP102 with device node /dev/tip102_1 and so on.

The allocation of device nodes to physical TIP102 modules depends on the search order of the IPAC carrier driver. Please refer to the IPAC carrier user manual.

Loading of the TIP102 device driver will only work if kernel KMOD support is installed, necessary carrier board drivers already installed and the kernel dependency file is up to date. If KMOD support isn't available you have to build either a new kernel with KMOD installed or you have to install the IPAC carrier kernel modules manually in the correct order (please refer to the IPAC carrier driver user manual).

2.4 Remove device driver from the running kernel

- To remove the device driver from the running kernel login as root and execute the following command:

  # modprobe tip102drv –r

If your kernel has enabled devfs, all /dev/tip102_... nodes will be automatically removed from your file system after this.

Be sure that the driver isn't opened by any application program. If opened you will get the response "*tip102drv: Device or resource busy*" and the driver will still remain in the system until you close all opened files and execute *modprobe –r* again.

2.5 Change Major Device Number

The TIP102 driver use dynamic allocation of major device numbers by default. If this isn't suitable for the application it's possible to define a major number for the driver. If the kernel has enabled devfs the driver will not use the symbol TIP102_MAJOR.

To change the major number edit the file tip102drv.c, change the following symbol to appropriate value and enter **make install** to create a new driver.

TIP102_MAJOR	Valid numbers are in range between 0 and 255. A value of 0 means dynamic number allocation.

Example:

```
#define TIP102_MAJOR        122
```

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 <fcntl.h>

int open (const char *filename, int flags)

DESCRIPTION

The open function creates and returns a new file descriptor for the file named by *filename*. The *flags* argument controls how the file is to be opened. This is a bit mask; you create the value by the bitwise OR of the appropriate parameters (using the | operator in C). See also the GNU C Library documentation for more information about the open function and open flags.

EXAMPLE

```
int fd;

…

fd = open(“/dev/tip102_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

ENODEV	The requested minor device does not exist.

This is the only error code returned by the driver, other codes may be returned by the I/O system during open. For more information about open error codes, see the *GNU C Library description – Low-Level Input/Output*.

SEE ALSO

GNU C Library description – Low-Level Input/Output

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

ENODEV	The requested minor device does not exist.

This is the only error code returned by the driver, other codes may be returned by the I/O system during close. For more information about close error codes, see the *GNU C Library description – Low-Level Input/Output*.

SEE ALSO

GNU C Library description – Low-Level Input/Output

3.3 ioctl()

NAME

ioctl() – device control functions

SYNOPSIS

#include <sys/ioctl.h>

int ioctl(int filedes, int request [, void *argp])

DESCRIPTION

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

The argument *request* specifies the control code for the operation. The optional argument *argp* depends on the selected request and is described for each request in detail later in this chapter.

The following ioctl codes are defined in *tip102.h*:

Symbol	Meaning
T102_IOCG_READ_ADC	Read ADC input value
T102_IOCS_WRITE_DAC	Set DAC output value
T102_IOCG_READ_COUNTER	Read actual counter value
T102_IOCS_COUNTER_PRESET	Set counter preset value
T102_IOCG_READ_REGISTER	Read counter register value
T102_IOCS_WRITE_REGISTER	Set counter register value
T102_IOCS_SET_REFMODE	Set reference mode
T102_IOCS_EVENT_WAIT	Wait for interrupt event

See behind for more detailed information on each control code.

To use these TIP102 specific control codes the header file tip102.h must be included in the application.

RETURNS

On success, zero is returned. In the case of an error, a value of –1 is returned. The global variable *errno* contains the detailed error code.

ERRORS

EINVAL	Invalid argument. This error code is returned if the requested ioctl function is unknown. Please check the argument request.

Other function dependant error codes will be described for each ioctl code separately. Note, the TIP102 driver always returns standard Linux error codes.

SEE ALSO

ioctl man pages

3.3.1 T102_IOCG_READ_ADC

NAME

T102_IOCG_READ_ADC - Read ADC input value

DESCRIPTION

This ioctl function attempts to read the actual analog input value from the TIP102 associated with the open file descriptor, *filedes*. The function starts a conversion waits (polled) and returns the result in the parameter buffer pointed to by *argp.*

The parameter buffer (*T102_ADC_DAC_BUFFER*) has the following layout:

```
typedef struct
{
        int             channel;
        short           data;
} T102_ADC_DAC_BUFFER, *PT102_ADC_DAC_BUFFER;
```

channel

This parameter specifies the channel number (axis) the ADC value should be read from. Allowed values are:

Value	Axis	Moduletypes
1	1	TIP102-1x, TIP102-2x
2	2	TIP102-2x

data

This parameter returns the result of the AD conversion. This is the value from the TIP102 register, that means only bit 15..4 are valid, bit 3..0 will always be set 0.

EXAMPLE

```
int fd;
int result;
T102_ADC_DAC_BUFFER adcBuf;

…

/* Read ADC from channel 1 */
adcBuf.channel = 1;

result = ioctl(fd, T102_IOCG_READ_ADC, &adcBuf);
if (result >= 0)
{
     /* OK */
     printf("Input Value %04Xh)\n", adcBuf.data);
}
else
{
     /* ERROR */
}

…
```

ERRORS

EFAULT	Invalid pointer to the read buffer. Invalid channel number specified.
ETIME	ADC conversion timed out.

SEE ALSO

ioctl man pages

3.3.2 T102_IOCS_WRITE_DAC

NAME

T102_IOCS_WRITE_DAC – Set the DAC output value

DESCRIPTION

This ioctl function sets the analog output value of the TIP102 associated with the open file descriptor, *filedes*. The value is supplied in the parameter buffer pointed to by *argp.*

The parameter buffer (*T102_ADC_DAC_BUFFER*) has the following layout:

```
typedef struct
{
        int             channel;
        short           data;
} T102_ADC_DAC_BUFFER, *PT102_ADC_DAC_BUFFER;
```

channel

This parameter specifies the channel number (axis) the DAC value should be set to. Allowed values are:

Value	Axis	Moduletypes
1	1	TIP102-1x, TIP102-2x
2	2	TIP102-2x

data

This parameter specifies the new DAC output value.

EXAMPLE

```
int fd;
int result;
T102_ADC_DAC_BUFFER dacBuf;

…

/* Set DAC value of channel 1 to 0x123*/
dacBuf.channel = 1;
dacBuf.data = 0x123;

result = ioctl(fd, T102_IOCS_WRITE_DAC, &dacBuf);
if (result >= 0)
{
     /* OK */
}
else
{
     /* ERROR */
}

…
```

ERRORS

EFAULT	Invalid pointer to the write buffer. Invalid channel number specified.

SEE ALSO

ioctl man pages

3.3.3 T102_IOCG_READ_COUNTER

NAME

T102_IOCG_READ_COUNTER - Read the actual counter value

DESCRIPTION

This ioctl function reads the actual counter value from the TIP102 associated with the open file descriptor, *filedes* and returns the value in the parameter buffer pointed to by *argp.*

The parameter buffer (*T102_COUNTER_BUFFER*) has the following layout:

```
typedef struct
{
        int             channel;
        long            data;
} T102_COUNTER_BUFFER, *PT102_COUNTER_BUFFER;
```

channel

This parameter specifies the channel number (axis). Allowed values are:

Value	Axis	Moduletypes
1	1	TIP102-1x, TIP102-2x
2	2	TIP102-2x

data

This parameter returns the actual value of the counter.

EXAMPLE

```
int fd;
int result;
T102_COUNTER_BUFFER countBuf;

…

/* Read counter of channel 2 */
countBuf.channel = 2;

result = ioctl(fd, T102_IOCG_READ_COUNTER, &countBuf);
if (result >= 0)
{
     /* OK */
     printf("Counter: %ld)\n", countBuf.data);
}
else
{
     /* ERROR */
}

…
```

ERRORS

EFAULT	Invalid pointer to the read buffer. Invalid channel number specified.

SEE ALSO

ioctl man pages

3.3.4 T102_IOCS_COUNTER_PRESET

NAME

T102_IOCS_COUNTER_PRESET – Set the counter preset value

DESCRIPTION

This ioctl function sets the actual counter preset value of the TIP102 associated with the open file descriptor, *filedes.* The value is supplied in the parameter buffer pointed to by *argp.*

The parameter buffer (*T102_COUNTER_BUFFER*) has the following layout:

```
typedef struct
{
        int             channel;
        long            data;
} T102_COUNTER_BUFFER, *PT102_COUNTER_BUFFER;
```

channel

This parameter specifies the channel number (axis). Allowed values are:

Value	Axis	Moduletypes
1	1	TIP102-1x, TIP102-2x
2	2	TIP102-2x

data

This parameter specifies the new preset value of the counter.

EXAMPLE

```
int fd;
int result;
T102_COUNTER_BUFFER countBuf;

…

/* Set counter preset value of channel 2 to 0x112233 */
countBuf.channel = 2;
countBuf.data = 0x112233;

result = ioctl(fd, T102_IOCS_COUNTER_PRESET, &countBuf);
if (result >= 0)
{
     /* OK */
}
else
{
     /* ERROR */
}

…
```

ERRORS

EFAULT	Invalid pointer to the read buffer. Invalid channel number specified.

SEE ALSO

ioctl man pages

3.3.5 T102_IOCG_READ_REGISTER

NAME

T102_IOCG_READ_REGISTER - Read value of a specified register

DESCRIPTION

This ioctl function reads the value of a specified register from the TIP102 associated with the open file descriptor, *filedes* and returns the value in the parameter buffer pointed to by *argp.*

The parameter buffer (*T102_REGISTER_BUFFER*) has the following layout:

```
typedef struct
{
        int             channel;
        int             regsel;
        unsigned char   data;
} T102_REGISTER_BUFFER, *PT102_REGISTER_BUFFER;
```

channel

This parameter specifies the channel number (axis). Allowed values are:

Value	Axis	Moduletypes
1	1	TIP102-1x, TIP102-2x
2	2	TIP102-2x

regsel

This parameter selects the register that should be read. The following, allowed register selections are defined in tip102.h:

Selection	Description
T102_INPSR	Selects the input status register
T102_OUTCR	Selects the output control register
T102_CNTDA	Selects the 8-bit counter data register
T102_CNTCS	Selects the counter control and status register
T102_CONCR	Selects the configuration control register

data

This parameter returns the actual value of the register.

EXAMPLE

```
int fd;
int result;
T102_REGISTER_BUFFER regBuf;

…

/* Read value from output control register of channel 2 */
regBuf.channel = 2;
regBuf.regsel = T102_OUTCR;

result = ioctl(fd, T102_IOCG_READ_REGISTER, &regBuf);
if (result >= 0)
{
     /* OK */
     printf("Register Value: %02Xh)\n", regBuf.data);
}
else
{
     /* ERROR */
}

…
```

ERRORS

EFAULT	Invalid pointer to the read buffer. Invalid channel number specified. Invalid register specified.

SEE ALSO

ioctl man pages

3.3.6 T102_IOCS_WRITE_REGISTER

NAME

T102_IOCS_WRITE_REGISTER – Writes a new value to a specified register

DESCRIPTION

This ioctl function sets the value of a specified register from the TIP102 associated with the open file descriptor, *filedes*. The register selection and value are supplied in the parameter buffer pointed to by *argp.*

The parameter buffer (*T102_REGISTER_BUFFER*) has the following layout:

```
typedef struct
{
        int             channel;
        int             regsel;
        unsigned char   data;
} T102_REGISTER_BUFFER, *PT102_REGISTER_BUFFER;
```

channel

This parameter specifies the channel number (axis). Allowed values are:

Value	Axis	Moduletypes
1	1	TIP102-1x, TIP102-2x
2	2	TIP102-2x

regsel

This parameter selects the register that should be set. The following, allowed register selections are defined in tip102.h:

Selection	Description
T102_INPSR	Selects the input status register
T102_OUTCR	Selects the output control register
T102_CNTDA	Selects the 8-bit counter data register
T102_CNTCS	Selects the counter control and status register
T102_CONCR	Selects the configuration control register

data

This parameter contains the new register value.

EXAMPLE

```
int fd;
int result;
T102_REGISTER_BUFFER regBuf;

…

/* Write value (0x12) to the output control register of channel 2 */
regBuf.channel = 2;
regBuf.regsel = T102_OUTCR;
regBuf.data = 0x12;

result = ioctl(fd, _IOCG_WRITE_REGISTER, &regBuf);
if (result >= 0)
{
     /* OK */
}
else
{
     /* ERROR */
}

…
```

ERRORS

EFAULT	Invalid pointer to the read buffer. Invalid channel number specified. Invalid register specified.

SEE ALSO

ioctl man pages

3.3.7 T102_IOCS_SET_REFMODE

NAME

T102_IOCS_SET_REFMODE – Setup reference mode

DESCRIPTION

This ioctl function sets up the reference mode of the specified channel on the TIP102 associated with the open file descriptor, *filedes*. The mode selection is supplied in the parameter buffer pointed to by *argp.*

The parameter buffer (*T102_REFMODE_BUFFER*) has the following layout:

```
typedef struct
{
    int         channel;
    int         refMode;
} T102_REFMODE_BUFFER, *PT102_REFMODE_BUFFER;
```

channel

This parameter specifies the channel number (axis). Allowed values are:

Value	Axis	Moduletypes
1	1	TIP102-1x, TIP102-2x
2	2	TIP102-2x

refMode

This parameter configures the reference mode. The following flags are defined in tip102.h and can be ORed (for detailed information refer to the TIP102 User Manual):

Selection	Description
T102_REF_SWITCH	Reference mode with reference switch
T102_EXT_TRIGGER	Reference mode with external trigger
T102_AUTO_REF	Auto reference

EXAMPLE

```
int fd;
int result;
T102_REFMODE_BUFFER refBuf;

…

/* Set auto reference mode for channel 2 */
refBuf.channel = 2;
refBuf.refMode = T102_AUTO_REF;

result = ioctl(fd, T102_IOCS_SET_REFMODE, &refBuf);
if (result >= 0)
{
     /* OK */
}
else
{
     /* ERROR */
}

…
```

ERRORS

EFAULT	Invalid pointer to the read buffer. Invalid channel number specified. Invalid flags specied.

SEE ALSO

ioctl man pages

3.3.8 T102_IOCS_EVENT_WAIT

NAME

T102_IOCS_EVENT_WAIT – Wait for interrupt event

DESCRIPTION

This ioctl function waits for an interrupt event on the TIP102 associated with the open file descriptor, *filedes*. The channel and timeout value are supplied in the parameter buffer pointed to by *argp*.

The parameter buffer (*T102_EVENT_BUFFER*) has the following layout:

```
typedef struct
{
    int         channel;
    int         timeout;
} T102_EVENT_BUFFER, *PT102_EVENT_BUFFER;
```

channel

This parameter specifies the channel number (axis). Allowed values are:

Value	Axis	Moduletypes
1	1	TIP102-1x, TIP102-2x
2	2	TIP102-2x

timeout

This parameter specifies the maximum time to wait for the interrupt event. If the specified time has expired, the function will return with an error. The timeout time is specified in ticks.

EXAMPLE

```
int fd;
int result;
T102_EVENT_BUFFER evBuf;

…

/* Wait for interrupt event on channel 2, timeout after 5000 ticks */
evBuf.channel = 2;
evBuf.timeout = 5000;

result = ioctl(fd, T102_IOCS_EVENT_WAIT, &evBuf);
if (result >= 0)
{
     /* OK */
}
else
{
     /* ERROR */
}

…
```

ERRORS

EFAULT	Invalid pointer to the read buffer. Invalid channel number specified.
ETIME	The interrupt has not occurred in the specified time.

SEE ALSO

ioctl man pages