The Embedded I/O Company

TEWS TECHNOLOGIES

TIP111-SW-95

QNX-Neutrino Device Driver

Motion Controller with Absolute Encoder Interface (SSI)

Version 1.0.x

User Manual

Issue 1.0.0

January 2007

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

TIP111-SW-95

QNX-Neutrino Device Driver

Motion Controller with Absolute Encoder Interface (SSI)

Supported Modules:
TIP111

Issue	Description	Date
1.0.0	First Issue	January 5, 2007

Table of Contents

1 Introduction

The TIP111-SW-95 QNX-Neutrino device driver allows the operation of the TIP111 Motion Controller IPAC on QNX-Neutrino systems and requires the TEWS QNX-Neutrino IPAC Carrier Driver Software (CARRIER-SW-95).

The TIP111 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.

The TIP111-SW-95 device driver supports the following features:

- Configure encoder interface
- Read encoder value
- Read ADC input value
- Write DAC output value
- Set/Clear digital outputs
- Read Status register

The TIP111-SW-95 device driver supports the modules listed below:

TIP111-1x	1 Channel Motion Controller (SSI)	IndustryPack® compatible
TIP111-2x	2 Channel Motion Controller (SSI)	IndustryPack® compatible

To get more information about the features and use of TIP111 devices it is recommended to read the manuals listed below.

TIP111 User manual

TIP111 Engineering Manual

CARRIER-SW-95 User Manual

2 Installation

Following files are located on the distribution media:

Directory path '.\TIP111-SW-95\':

TIP111-SW-95-SRC.tar.gz	GZIP compressed archive with driver source code
TIP111-SW-95-1.0.0.pdf	PDF copy of this manual
ChangeLog.txt	Release history
Release.txt	Release information

For installation the files have to be copied to the desired target directory.

The GZIP compressed archive TIP111-SW-95-SRC.tar.gz contains the following files and directories:

Directory path './tip111/':

driver/tip111drv.c	TIP111 device driver source
driver/tip111def.h	TIP111 driver include file
driver/tip111.h	TIP111 include file for driver and application
driver/Makefile	TIP111 driver makefile
example/tip111exa.c	Example application
example/common.mk	TIP111 example make parameters
example/Makefile	TIP111 example recursive makefile
example/nto/Makefile	TIP111 example recursive makefile
example/nto/x86/Makefile	TIP111 example recursive makefile
example/nto/x86/o/Makefile	TIP111 example recursive makefile

In order to perform an installation, copy TIP111-SW-95.tar.gz to */usr/src* and extract all files of the archive. (`tar –xzf TIP111-SW-95.tar.gz`)

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 is 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 distribution media.

Its absolute important to extract the TIP111-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/tip111/driver* directory

Execute the Makefile

```
# make install
```

After successful completion the driver binary will be installed in the */bin* directory.

2.2 Build the example application

Change to the /usr/src/tip111/example directory

Execute the Makefile

```
# make install
```

After successful completion the example binary (*tip111exa*) will be installed in the */bin* directory.

2.3 Start the driver process

To start the TIP111 Resource Manager (Device Driver) you only have to start the TEWS TECHNOLOGIES IPAC Carrier Driver. The Carrier Driver automatically detects installed TEWS IPACs and dynamically loads the concerning module(s).

The TIP111 Resource Manager registers a device for each TIP111 in the QNX-Neutrinos pathname space under following name, where *n* specifies the used IPAC slot number (please refer to the IPAC Carrier Driver Manual):

```
/dev/tip111_n
```

This pathname must be used in the application program to open a path to the desired TIP111 device.

For debugging you can start the IPAC Carrier Driver with the –V (verbose) option. Now the Resource Manager will print versatile information about TIP111 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 a TIP111 device.

PARAMETER

pathname

Specifies the device to open.

flags

Controls how the file is to be opened. TIP111 devices must be opened *O_RDWR*.

EXAMPLE

```
int  fd;

…

fd = open("/dev/tip111_0", O_RDWR);
if (fd == -1)
{
     /* Handle error */
}
```

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.

ERROR CODES

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 a file.

PARAMETER

filedes

Specifies the file to close.

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.

ERROR CODES

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.

PARAMETER

filedes

Specifies the device to perform the requested operation.

dcmd

Specifies the control code for the operation. The following commands are defined (tip111.h):

Command	Description
DCMD_TIP111_READ_ENC	Read current encoder value
DCMD_TIP111_READ_ADC	Read ADC value
DCMD_TIP111_WRITE_DAC	Write DAC value
DCMD_TIP111_READ_STATUS	Read input status register
DCMD_TIP111_SET_OUTPUT	Set output control bit(s)
DCMD_TIP111_CLR_OUTPUT	Clear output control bit(s)
DCMD_TIP111_CONFIG	Configure encoder interface

data_ptr

Depends on the command and will be described for each command in detail later in this chapter. Usually points to a buffer that passes data between the user task and the driver

n_bytes

Depends on the command and will be described for each command in detail later in this chapter. Usually defines the size of the buffer pointed by *data_ptr*.

dev_info_ptr

Is unused for the TIP111 driver and should be set to *NULL*.

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 *dcmd*.

Other function dependant error codes will be described for each *devctl()* code separately. Note, the TIP111 driver always returns standard QNX error codes.

SEE ALSO

Library Reference - devctl()

3.3.1 DCMD_TIP111_READ_ENC

DESCRIPTION

This function reads the current position value from the connected SSI encoder. The data transfer (serial) could take up to 480 microseconds depending on the number of data bits and the selected clock rate. During the data transfer the driver is in a busy loop (no interrupt notification) and blocks other tasks. To minimize the blocking time please use always the maximal possible transfer rate (clock rate).

Be aware about the fact that a position jump occurs if the position bounds will be crossed (e.g. 24-bit SSI encoder will jump for 0 to 16777215 and vise versa from 16777215 to 0).

Before reading encoder values the encoder interface must be configured with the *devctl()* function code *DCMD_TIP111_CONFIG*.

The function specific argument *data_ptr* points to an encoder data structure (*TIP111_ENC_BUF*) and *n_bytes* specify its length in bytes.

```
typedef struct
{
        int             channelNo;
        unsigned long   value;
} TIP111_ENC_BUF;
```

channelNo

Specifies the channel number (axis). Allowed channel numbers are 1 and 2 for TIP111-2x with two channels or only 1 for TIP111-1x with one channel

value

This field returns the encoder value.

EXAMPLE

```
int                fd;
int                result;
TIP111_ENC_BUF     encBuf;

…

/* Get encoder value of channel 1 */
encBuf.channelNo = 1;

result = devctl(fd, DCMD_TIP111_READ_ENC, &encBuf, sizeof(encBuf), NULL);
if (result != EOK)
{
     /* Handle error */
}
else
{
     printf("Encoder: %ld\n", encBuf.value);
}
```

ERRORS

EINVAL	Invalid argument specified (e.g. channel number).
EIO	Parity error occurred during transmission.
ETIMEDOUT	Timeout occurred during data transfer from the encoder.
EACCES	The encoder interface has not been configured.

3.3.2 DCMD_TIP111_READ_ADC

DESCRIPTION

This function starts an AD conversion and returns the converted analog value (2's complement) to the caller.

The function specific argument *data_ptr* points to an analog value data structure (*TIP111_ADC_DAC_BUF*) and *n_bytes* specify its length in bytes.

```
typedef struct
{
        int             channelNo;
        short           value;
} TIP111_ADC_DAC_BUF;
```

channelNo

Specifies the channel number (axis). Allowed channel numbers are 1 and 2 for TIP111-2x with two channels or only 1 for TIP111-1x with one channel

value

This field returns the result of the AD conversion. The value range will be from -2048 for (~-10V) up to 2047 for (~+10V).

EXAMPLE

```
int                 fd;
int                 result;
TIP111_ADC_DAC_BUF adcBuf;

…

/* Get ADC value of channel 1 */
adcBuf.channelNo = 1;

result = devctl(fd, DCMD_TIP111_READ_ADC, &adcBuf, sizeof(adcBuf), NULL);
if (result != EOK)
{
     /* Handle error */
}
else
{
     printf("ADC: %ld\n", adcBuf.value);
}
```

ERRORS

EINVAL	Invalid argument specified (e.g. channel number).
ETIMEDOUT	Timeout occurred during AD conversion.

3.3.3 DCMD_TIP111_WRITE_DAC

DESCRIPTION

This function starts a DAC conversion with a specified value.

The function specific argument *data_ptr* points to an analog value data structure (*TIP111_ADC_DAC_BUF*) and *n_bytes* specify its length in bytes.

```
typedef struct
{
        int             channelNo;
        short           value;
} TIP111_ADC_DAC_BUF;
```

channelNo

Specifies the channel number (axis). Allowed channel numbers are 1 and 2 for TIP111-2x with two channels or only 1 for TIP111-1x with one channel

value

This value specifies the (16-bit) DAC output value to be used. The value range is from -32768 for (~-10V) up to 32767 for (~+10V).

EXAMPLE

```
int                 fd;
int                 result;
TIP111_ADC_DAC_BUF dacBuf;

…

/* Set output of channel 1 to ~5V */
dacBuf.channelNo = 1;
dacBuf.value = 16384;

result = devctl(fd, DCMD_TIP111_WRITE_DAC, &dacBuf, sizeof(dacBuf), NULL);
if (result != EOK)
{
     /* Handle error */
}
```

ERRORS

EINVAL	Invalid argument specified (e.g. channel number).

3.3.4 DCMD_TIP111_READ_STATUS

DESCRIPTION

This function reads the input status register (INPSR).

The function specific argument *data_ptr* points to an I/O data structure (*TIP111_IO_BUF*) and *n_bytes* specify its length in bytes.

```
typedef struct
{
        int             channelNo;
        unsigned char   value;
} TIP111_IO_BUF;
```

channelNo

Specifies the channel number (axis). Allowed channel numbers are 1 and 2 for TIP111-2x with two channels or only 1 for TIP111-1x with one channel

value

This field returns the current content of the status register (INPSR). The following bits are defined in tip111.h:

Definition	Value	Description
TIP111_LOW_LIMIT	(1<<0)	State of low limit switch input pin
TIP111_HIGH_LIMIT	(1<<1)	State of high limit switch input pin
TIP111_INPUT_1	(1<<2)	State of reference input pin
TIP111_EXT_TRIG_IN	(1<<3)	State of trigger input pin
TIP111_INPUT_2	(1<<4)	State of general input pin
TIP111_PARITY_ERR	(1<<5)	Specifies an encoder parity error
TIP111_DATA_VALID	(1<<6)	Specifies encoder data ready

EXAMPLE

```
int                 fd;
int                 result;
TIP111_IO_BUF       statBuf;

…

/* Get state of channel 1 */
statBuf.channelNo = 1;

result = devctl(fd, DCMD_TIP111_READ_STATUS,
                &statBuf, sizeof(statBuf), NULL);
if (result != EOK)
{
     /* Handle error */
}
else
{
     printf("State: %ld\n", statBuf.value);
}
```

ERRORS

EINVAL	Invalid argument specified (e.g. channel number).

3.3.5 DCMD_TIP111_SET_OUTPUT

DESCRIPTION

This function sets a specified set of output pins.

The function specific argument *data_ptr* points to an I/O data structure (*TIP111_IO_BUF*) and *n_bytes* specify its length in bytes.

```
typedef struct
{
        int             channelNo;
        unsigned char   value;
} TIP111_IO_BUF;
```

channelNo

Specifies the channel number (axis). Allowed channel numbers are 1 and 2 for TIP111-2x with two channels or only 1 for TIP111-1x with one channel

value

This value is a mask specifying the output bits that shall be activated. If more than one output shall be activated, the bits can be ORed bitwise. The following output bits are defined:

Definition	Value	Description
TIP111_SERVO_ENA	(1<<0)	Sets servo amplifier enable output
TIP111_STATUS_LED	(1<<1)	Sets status LED output
TIP111_EXT_TRIG_OUT	(1<<2)	Set external trigger output

EXAMPLE

```
int                fd;
int                result;
TIP111_IO_BUF      setBuf;

…

/* Enable status LED output of channel 1 */
setBuf.channelNo = 1;
setBuf.value = TIP111_STATUS_LED;

…

result = devctl(fd, DCMD_TIP111_SET_OUTPUT, &setBuf, sizeof(setBuf), NULL);
if (result != EOK)
{
     /* Handle error */
}
```

ERRORS

EINVAL	Invalid argument specified (e.g. channel number).

3.3.6 DCMD_TIP111_CLR_OUTPUT

DESCRIPTION

This function clears a specified set of output pins.

The function specific argument *data_ptr* points to an I/O data structure (*TIP111_IO_BUF*) and *n_bytes* specify its length in bytes.

```
typedef struct
{
        int             channelNo;
        unsigned char   value;
} TIP111_IO_BUF;
```

channelNo

Specifies the channel number (axis). Allowed channel numbers are 1 and 2 for TIP111-2x with two channels or only 1 for TIP111-1x with one channel

value

This value is a mask specifying the output bits that shall be deactivated. If more than one output shall be deactivated, the bits can be ORed bitwise. The following output bits are defined:

Definition	Value	Description
TIP111_SERVO_ENA	(1<<0)	Sets servo amplifier enable output
TIP111_STATUS_LED	(1<<1)	Sets status LED output
TIP111_EXT_TRIG_OUT	(1<<2)	Set external trigger output

EXAMPLE

```
int                fd;
int                result;
TIP111_IO_BUF      clrBuf;

…

/* Disable servo amplifier output of channel 1 */
clrBuf.channelNo = 1;
clrBuf.value = TIP111_SERVO_ENA;

…

result = devctl(fd, DCMD_TIP111_CLR_OUTPUT, &clrBuf, sizeof(clrBuf), NULL);
if (result != EOK)
{
     /* Handle error */
}
```

ERRORS

EINVAL	Invalid argument specified (e.g. channel number).

3.3.7 DCMD_TIP111_CONFIG

DESCRIPTION

This function configures the encoder interface of the specified channel.

This function must be called to configure the encoder interface before any encoder data can be read.

The function specific argument *data_ptr* points to a configuration data structure (*TIP111_CONFIG_BUF*) and *n_bytes* specify its length in bytes.

```
typedef struct
{
        int             channelNo;
        int             numDataBits;
        int             clockRate;
        int             enableParity;
        int             enableGrayCode;
} TIP111_CONFIG_BUF;
```

channelNo

Specifies the channel number (axis). Allowed channel numbers are 1 and 2 for TIP111-2x with two channels or only 1 for TIP111-1x with one channel

numDataBits

Specifies the number of data bits of the connected absolute encoder. Valid values are between 1 and 32 data bits.

clockRate

Specifies the clock speed of the serial data transfer. The clock speed can be programmed in steps of 1 microsecond in the range from 1 to 15.

enableParity

Set *TRUE* to enable parity or *FALSE* to disable parity. Enable parity only if this feature is supported by the encoder, otherwise *DCMD_TIP111_READ_ENC* will return a parity error (*EIO*).

enableGrayCode

Set *TRUE* to enable Gray Code for serial data transfer or *FALSE* for Binary Code. This configuration must match to the encoder properties; otherwise strange position values will be transferred. Please note the driver cannot make any plausibility check and therefore no errors will be returned if the configuration do not match.

EXAMPLE

```
int                fd;
int                result;
TIP111_CONFIG_BUF  confBuf;

…

/* Set up encoder interfac of channel 1 */
confBuf.channelNo = 1;
confBuf.numDataBits = 24;        /* Databits:  24bit */
confBuf.enableParity = FALSE;    /* Paritybit: None */
confBuf.enableGrayCode = TRUE;   /* Datacoding:GrayCode */
confBuf.clockRate = 2;           /* Clockrate: 2us */

result = devctl(fd, DCMD_TIP111_CONFIG,&confBuf, sizeof(confBuf), NULL);
if (result != EOK)
{
     /* Handle error */
}
```

ERRORS

EINVAL	Invalid argument specified (e.g. channel number).