The Embedded I/O Company

TEWS TECHNOLOGIES

TIP501-SW-95

QNX-Neutrino Device Driver

Optically Isolated 16 Channel 16 Bit ADC

Version 1.1.x

User Manual

Issue 1.1.0

October 2009

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

TIP501-SW-95

QNX-Neutrino Device Driver

Optically Isolated 16 Channel 16 Bit ADC

Supported Modules:
TIP501

Issue	Description	Date
1.0	First Issue	October 13, 2004
1.1.0	Revision	October 19, 2009

Table of Contents

1 Introduction

The TIP501-SW-95 QNX-Neutrino device driver allows the operation of a TIP501 Optically Isolated 16 Channel 16 Bit ADC IP on QNX-Neutrino operating systems.

The TIP501 device driver is basically implemented as a user installable Resource Manager and started by the TEWS IPAC Carrier Driver (CARRIER-SW-95) if a TIP501 module was found during scanning of supported carrier boards.

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 TIP501-SW-95 device driver supports the following features:

- Reading analog input values from specified channels
- Standard and pipeline mode
- Single ended and differential modes
- Automatic offset and gain correction with factory calibration data

The TIP501-SW-95 device driver supports the modules listed below:

TIP501-10	Optically isolated 16 channel 16 bit ADC input voltage range +/-10V, gain 1, 2, 5, 10	(IndustryPack®)
TIP501-11	Optically isolated 16 channel 16 bit ADC input voltage range +/-10V, gain 1, 2, 4, 8	(IndustryPack®)
TIP501-20	Optically isolated 16 channel 16 bit ADC input voltage range 0V to +10V, gain 1, 2, 5, 10	(IndustryPack®)
TIP501-21	Optically isolated 16 channel 16 bit ADC input voltage range 0V to +10V, gain 1, 2, 4, 8	(IndustryPack®)

To get more information about the features and use of TIP501 devices it is recommended to read the manuals listed below.

TIP501 User manual

TIP501 Engineering Manual

CARRIER-SW-95 User Manual

1.1 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 other differences. Also, the varying byte ordering (big-endian versus little-endian) of CPU boards will cause problems when accessing the IndustryPack I/O and memory spaces.

To simplify the implementation of IPAC device drivers which should work with every 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-95 is part of this TIP501-SW-95 distribution. It is located in the directory CARRIER-SW-95 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-95 User Manual for a detailed description on how to install and setup the CARRIER-SW-95 device driver, and for a description of the TEWS TECHNOLOGIES IPAC Carrier Driver concept.

2 Installation

Following files are located on the distribution media:

Directory path 'TIP501-SW-95':

TIP501-SW-95-SRC.tar.gz	GZIP compressed archive with driver source code
TIP501-SW-95-1.1.0.pdf	PDF copy of this manual
ChangeLog.txt	Release history
Release.txt	Release information

The GZIP compressed archive TIP501-SW-95-SRC.tar.gz contains the following files and directories:

Directory path 'tip501':

driver/tip501.c	Device driver source
driver/tip501.h	Device driver and application include file
driver/tip501def.h	Device driver include file
driver/Makefile	Recursive multiplatform build tree
driver/common.mk	
driver/nto/Makefile	
driver/nto/x86/Makefile	
driver/nto/x86/dll/Makefile	
example/tip501exa.c	Example application
example/Makefile	Recursive multiplatform build tree
example/common.mk	
example/nto/Makefile	
example/nto/x86/Makefile	
example/nto/x86/o/Makefile	

For installation, copy the tar-archive TIP501-SW-95-SRC.tar.gz to /usr/src and extract all files (e.g tar -xzvf TIP501-SW-95-SRC.tar.gz). Afterwards, the necessary directory structure for the automatic build and the source files are available underneath the new directory called tip501.

Change to the driver directory */usr/src/tip501/driver* and copy the header file *tip501.h* to */usr/include* allowing user application programs sharing the TIP501 driver interface definitions and data structures.

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 distribution media.

It is very important to extract the TIP501-SW-95-SRC.tar.gz in the /usr/src directory, because otherwise the automatic build with make will fail.

2.1 Build the device driver

Change to the directory */usr/src/tip501/driver* and execute the Makefile

```
# make install
```

After successful completion the driver dynamic library *tip501.so* will be installed in the directory */lib/dll*.

2.2 Build the example application

Change to the directory */usr/src/tip501/example* and execute the Makefile

```
# make install
```

After successful completion the example binary *tip501exa* will be installed in the directory */bin*.

2.3 Start the driver process

To start the TIP501 resource manager you have to start the TEWS TECHNOLOGIES IPAC carrier driver. The IPAC carrier driver detects installed TEWS IPAC modules automatically and loads the appropriate driver dynamic libraries.

```
# ipac_class &
```

The TIP501 resource manager registers a device for each TIP501 in the QNX-Neutrinos pathname space. The device file /dev/tip501_0 belongs to the first TIP501 found the device file /dev/tip501_1 to the second TIP501 and so forth (please refer to the IPAC carrier driver manual for detailed information of the module search order)

This device file must be used in the application program to open a path to the desired TIP501 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 TIP501 configuration and command execution on the terminal window. For further details about debugging, please see the 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 TIP501 named by *pathname.* The flags argument controls how the file is to be opened. TIP501 devices must be opened *O_RDWR*.

EXAMPLE

```
int  fd;

fd = open(“/dev/tip501_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.

ERRORS

Returns only QNX-Neutrino specific error codes, see QNX-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 QNX-Neutrino specific error codes, see QNX-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 TIP501 driver and should be set to NULL.

The following devctl command codes are defined in *tip501.h* :

Value	Meaning
DCMD_TIP501_READ	Read ADC channel(s)
DCMD_TIP501_GET_INFO	Get device information

See behind for more detailed information on each control code.

To use these TIP501 specific control codes the header file tip501.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

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

Other function dependent error codes will be described for each *devctl()* code separately. Note, the TIP501 driver always returns standard QNX Neutrino error codes.

SEE ALSO

Library Reference - devctl()

3.3.1 DCMD_TIP501_READ

NAME

DCMD_TIP501_READ – Read ADC channel(s)

DESCRIPTION

The read function starts an AD conversion at the specified input channel of the TIP501 associated with the file descriptor *filedes*. A pointer to the read buffer structure (*T501_READ_BUFFER*) is passed by read function argument *buffer* to the driver. The argument *size* specifies the length of the read buffer and must be set to the length of the structure *T501_READ_BUFFER*.

Before calling the read function some elements of the read buffer must be set to appropriate values (see below for a detailed description of each element). After successful execution the element *data* returns the converted analog input value as a two's complement integer value.

The read function will always use the fastest possible operating mode.

```
typedef struct
{
    int     chan;
    int     gain;
    int     flags;
    int     data;
} T501_READ_BUFFER;
```

chan

This parameter specifies the channel number at which the conversion will be started. Valid channel numbers are 1...16 if single-ended is selected or 1...8 for differential input.

gain

This parameter specifies the gain for the input voltage amplifier. Valid gain constants are listed below:

Definition	Gain	Valid for TIP501 Variant
T501_GAIN_1	1	TIP501-10/-11/-20/-21
T501_GAIN_2	2	TIP501-10/-11/-20/-21
T501_GAIN_4	4	TIP501-11/-21
T501_GAIN_5	5	TIP501-10/-20
T501_GAIN_8	8	TIP501-11/-21
T501_GAIN_10	10	TIP501-10/-20

flags

This bit mask controls the read operation; you create the value by the bitwise OR of the appropriate parameters (using the | operator in C).

T501_DIFF	Use differential analog inputs. If this flag is omitted single-ended inputs will be selected.
T501_PIPELINE	Use data pipeline mode. In this mode converted data from the previous conversion will be read and a new conversion is started. If this flag is omitted, data from the new started conversion will be read (see also the TIP501 User Manual for more information).
T501_CORRECTION	Perform an automatic offset and gain correction with factory calibration data stored in the TIP501 ID-PROM. If this flag is omitted the converted data will be read directly.

data

This parameter returns the converted analog input value (two's complement).

EXAMPLE

```
int  fd;
int  result;
T501_READ_BUFFER  rdBuf;

rdBuf.chan    = 1;
rdBuf.gain    = T501_GAIN_1;
rdBuf.flags   = T501_DIFF | T501_CORRECTION;

result = devctl(fd, DCMD_TIP501_READ, &rdBuf, sizeof(rdBuf), NULL);

if (result != EOK) {
     // process error;
}
```

RETURNS

On success DCMD_TIP501_READ returns EOK. In the case of an error, the appropriate error code is returned by the function (not in errno!).

ERRORS

EINVAL	This error code is returned if the size of the buffer is wrong, or the gain or channel number is out of range.
ETIME	Timeout during AD conversion.

SEE ALSO

Library Reference - devctl()

3.3.2 DCMD_TIP501_GET_INFO

NAME

DCMD_TIP501_GET_INFO - Get device information

DESCRIPTION

This devctl function reads the module variant and the factory calibration data from the specified device and returns this information in the *T501_INFO_BUFFER* structure to the caller.

A pointer to the *T501_INFO_BUFFER* structure is passed by the parameter *data_ptr* to the driver.

```
typedef struct
{
        int     variant;
        short   offset_corr[4];
        short   gain_corr[4];
} T501_INFO_BUFFER;
```

variant

This parameter returns the module variant.

Value	Module Variant
10	TIP501-10
11	TIP501-11
20	TIP501-20
21	TIP501-21

offset_corr

The factory programmed correction data for offset correction is returned in this array. The index of the array specifies the input gain. (See table below)

gain_corr

The factory programmed correction data for gain correction is returned in this array. The index of the array specifies the input gain. (See table below)

Index	Gain (TIP501-10/20)	Gain (TIP501-11/21)
0	1	1
1	2	2
2	5	4
3	10	8

EXAMPLE

```
int  fd;
int  result;
T501_INFO_BUFFER  infoBuf;

result = devctl(fd, DCMD_TIP501_GET_INFO, &infoBuf, sizeof(infoBuf), NULL);

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

ERRORS

EFAULT	Invalid pointer to the T501_INFO_BUFFER. Please check the argument data_ptr.

SEE ALSO

Library Reference - devctl()