TIP500-SW-95
QNX-Neutrino Device Driver
TIP500 – 16/8 Channel 12 Bit ADC on SBS PCI40 Carrier

Version 1.0.x

Reference Manual
Issue 1.0

May 2002

TEWS TECHNOLOGIES GmbH
Am Bahnhof 7
D-25469 Halstenbek
Germany
Tel.: +49 (0)4101 4058-0
Fax.: +49 (0)4101 4058-19
http://www.tews.com
e-mail: info@tews.com

TIP500-SW-95
16/8 Channel 12 Bit ADC
QNX-Neutrino Device Driver

TEWS TECHNOLOGIES has made any effort to ensure that this manual is accurate and complete. However TEWS TECHNOLOGIES reserves the right to change the product described in this document at any time without notice.

This product has been designed to operate with IndustryPack® compatible carriers. Connection to incompatible hardware is likely to cause serious damage.

TEWS TECHNOLOGIES is not liable for any damage arising out of the application or use of the device described herein.

Issue	Description	Date
1.0	First Issue	May 27, 2002

Table of Contents

1 Introduction

The TIP500-SW-95 QNX-Neutrino device driver allows the operation of a TIP500 16/8 Channel 12 Bit ADC IP on QNX-Neutrino operating systems.

The TIP500 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:

- Read ADC value from specified input Channel
- Read module parameter
- Use factory programmed correction data for ADC correction

This driver will need a initializing of the carrier board, (e.g. SBS-PCI40). This driver should also announce the physical base addresses of the IP-slots. An example using the SBS-PCI40 is attached to the driver. This initialization software must be run before the driver is started.

2 Installation

The software is delivered on a PC formatted 3½" HD diskette.

Following driver specific files are located on the diskette:

/driver/tip500.c	Driver source code
/driver/tip500.h	Driver interface definitions and data structures
/driver/tip500def.h	Device driver include
/driver/node.h	Queue management definitions
/driver/node.c	Queue management source code
/example/example.c	Example application
/pci40/*	SBS-PCI40 installation example
TIP500-SW-95.pdf	This manual in PDF format

For installation create a new directory (e.g. *../tip500*) in the */usr/src* directory and copy the complete */driver* and */example* directories (with sub-directories and all files) from the distribution diskette into the new created project directory.

Note

It's absolute important to create the tip500 project directory in the /usr/src directory otherwise the automatic build with make will fail.

2.1 Build the device driver

1. Change to the */usr/src/tip500/driver* directory
2. Execute the Makefile

```
# make install
```

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

2.2 Build the example application

1. Change to the */usr/src/tip500/example* directory
2. Execute the Makefile

```
# make install
```

After successful completion the example binary (*t500exam*) will be installed in the */bin* directory.

2.3 Build the carrier board initialization example

1. Change to the */usr/src/pci40* directory
2. Execute the Makefile

```
# make install
```

After successful completion the example binary (*pci40*) will be installed in the */bin* directory.

2.4 Start the driver process

The carrier board initialization must be called before the driver is started. For example call the SBS-PCI40 initialization.

```
pci40
```

This initialization will printout the base addresses of I/O-, memory space and interrupt vector for each IP-slot.

To start the TIP500 device driver respective the TIP500 resource manager you have to enter the process name with optional parameter from the command shell or in the startup script.

```
tip500 –A<IOaddress> &
```

This will start the TIP500 resource manager with one module mounted at the specified <*IOaddress*>. (The address depends on the system, this address is printed out by the SBS-PCI40 initialization example).
For starting the TIP500 resource manager with more than one module, you have simply to add the additional IO-addresses behind the *–A* flag.

```
tip500 –A<IOaddress_0>,<IOaddress_1>,…,<IOaddress_n> &
```

The TIP500 Resource Manager registers created devices in the Neutrinos pathname space under following names.

```
/dev/tip500_0
/dev/tip500_1
…
/dev/tip500_x
```

This pathname must be used in the application program to open a path to the desired TIP500 device.

```
fd = open(“/dev/tip500_0”, O_RDWR);
```

For debugging you can start the TIP500 Resource Manager with the –v option. Now the Resource Manager will print versatile information about TIP500 configuration and command execution on the terminal window.

```
tip500 –v –A<IOaddress> &
```

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 TIP500 named by *pathname.*
The flags argument controls how the file is to be opened. TIP500 devices must be opened *O_RDWR*.

EXAMPLE

```
int    fd;

fd = open("/dev/tip500_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 TIP500 driver and should be set to NULL.

The following devctl command codes are defined in *TIP500.h* :

Value	**Meaning**
DCMD_T500_READ	Read ADC Input Channel
DCMD_T500_PARAM	Read Module Parameters

See behind for more detailed information on each control code.

Note

To use these TIP500 specific control codes the header file TIP500.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 *dcmd*.

Other function dependant error codes will be described for each devoctl code separately. Note, the TIP500 driver always returns standard QNX error codes.

SEE ALSO

Library Reference - devctl()

3.3.1 DCMD_T500_READ

NAME

DCMD_T500_READ - Read from ADC Input Channel

DESCRIPTION

This devctl function reads the actual value of the specified input channel. A pointer to the callers read buffer (*T500_READ_BUF*) and the size of this structure is passed by the parameters *data_ptr* and *n_bytes* to the device.

The *T500_READ_BUF* structure has the following layout:

```
typedef struct
{
    /* INPUT: */
    unsigned long    channel;        /* channel number: Sngl:1-16 or Diff:1-8      */
    unsigned long    flags;          /* TIP500_FL_CORR | TIP500_FL_DIFF            */
    unsigned long    gain;           /* TIP500_GAINx                               */

    /* OUTPUT: */
    int              data;           /* returned ADC value                         */
} T500_READ_BUF, *PT500_READ_BUF;
```

channel
Specifies the ADC input channel. If single-ended mode is used, channel numbers between 1 and 16 are allowed. If differential mode is used the channel number must be between 1 and 8.

flags
This argument specifies special options and the predefined values can be ORed.

TIP500_FL_CORR	Enable data correction
TIP500_FL_DIFF	Select differential input

gain
This argument specifies the input gain. The following table shows the allowed values.

TIP500_GAIN1	Input gain is 1
TIP500_GAIN2	Input gain is 2
TIP500_GAIN4	Input gain is 4 (only TIP500-11/-21)
TIP500_GAIN5	Input gain is 5 (only TIP500-10/-20
TIP500_GAIN8	Input gain is 8 (only TIP500-11/-21)
TIP500_GAIN10	Input gain is 10 (only TIP500-10/-20

data
This value will be filled with the actual input value. The value will be between –2048 and 2047 for TIP500-10/-11 and between 0 and 4095 for TIP500-20/-21

EXAMPLE

```
int              fd;
int              result;
T500_READ_BUF    ReadBuf;

…

/* Read ADC channel 1, gain=5, with corrected data */
ReadBuf.channel  = 1;
ReadBuf.flags    = TIP500_FL_CORR;
ReadBuf.gain     = TIP500_GAIN5;
result = devctl  (fd,
                  DCMD_T500_READ,
                  &ReadBuf,
                  sizeof(ReadBuf),
                  NULL);
if (result == EOK)
{
    /* ADC channel read successful */
}

…
```

ERRORS

EINVAL	Invalid argument. This error code is returned if either the size of the message buffer is too small, or the specified receive queue is out of range.
ETIMEDOUT	The conversion timed out, check the hardware.

SEE ALSO

Library Reference - devctl()

3.3.2 DCMD_T500_PARAM

NAME

DCMD_T500_PARAM - Read from module parameter

DESCRIPTION

This devctl function reads the actual value the specified input channel. A pointer to the callers parameter buffer (*T500_PARAM_BUF*) and the size of this structure is passed by the parameters *data_ptr* and *n_bytes* to the device.

The *T500_PARAM_BUF* structure has the following layout:

```
typedef struct
{
   /* OUTPUT: */
   unsigned long    modeltype;          /* returns modeltype (TIP500-xx) */
   int              ADC_offset_corr[4]; /* ADC offset correction data */
   int              ADC_gain_corr[4];   /* ADC gain correction data */
} T500_PARAM_BUF, *PT500_PARAM_BUF;
```

modeltype
This argument returns the model type of the TIP500.

ADC_offset_corr[]
ADC_gain_corr[]
These arrays return the correction data of the TIP500 for input. These values will be used by the driver if data correction is enabled for the read function.

EXAMPLE

```
int               fd;
int               result;
T500_PARAM_BUF    ParamBuf;

…

/* Read module parameters */
result = devctl  (fd,
                  DCMD_T500_PARAM,
                  &ParamBuf,
                  sizeof(ParamBuf),
                  NULL);
if (result == EOK)
{
     /* Parameter read successful */
}

…
```

ERRORS

EINVAL	Invalid argument. This error code is returned if either the size of the message buffer is too small, or the specified receive queue is out of range.

SEE ALSO

Library Reference - devctl()