



TIP550-SW-95
QNX-Neutrino Device Driver
TIP550 – 8(4) Channel 12 Bit DAC
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

TIP550-SW-95

8(4) Channel 12 Bit DAC

QNX-Neutrino Device Driver

This document contains information, which is proprietary to TEWS TECHNOLOGIES. Any reproduction without written permission is forbidden.

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.

IndustryPack is a registered trademark of GreenSpring Computers, Inc

©2002 by TEWS TECHNOLOGIES GmbH

Issue	Description	Date
1.0	First Issue	28. May 2002

Table of Contents

1	INTRODUCTION	4
2	INSTALLATION.....	5
2.1	Build the device driver	5
2.2	Build the example application	5
2.3	Build the carrier board initialization example	5
2.4	Start the driver process	6
2.5	Configure module.....	6
3	DEVICE INPUT/OUTPUT FUNCTIONS	7
3.1	open()	7
3.2	close()	8
3.3	devctl()	9
3.3.1	DCMD_T550_WRITE	11
3.3.2	DCMD_T550_PARAM.....	13
3.3.3	DCMD_T550_VRGCONFIG	15

1 Introduction

The TIP550-SW-95 QNX-Neutrino device driver allows the operation of a TIP550 - 8(4) Channel 12 Bit DAC IP on QNX-Neutrino operating systems.

The TIP550 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:

- Write DAC value to output Channel
- Read module parameter
- Use factory programmed correction data for DAC correction
- Configure output ranges

This driver will need a initialization 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/tip550.c	Driver source code
/driver/tip550.h	Driver interface definitions and data structures
/driver/tip550def.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
TIP550-SW-95.pdf	This manual in PDF format

For installation create a new directory (e.g. *../tip550*) 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 tip550 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/tip550/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/tip550/example* directory
2. Execute the Makefile

```
# make install
```

After successful completion the example binary (*t550exam*) 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 TIP550 device driver respective the TIP550 resource manager you have to enter the process name with optional parameter from the command shell or in the startup script.

```
tip550 -A<IOaddress> &
```

This will start the TIP550 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 TIP550 resource manager with more than one module, you have simply to add the additional IO-addresses behind the *-A* flag.

```
tip550 -A<IOaddress_0>,<IOaddress_1>,...,<IOaddress_n> &
```

The TIP550 Resource Manager registers created devices in the Neutrinos pathname space under following names.

```
/dev/tip550_0  
/dev/tip550_1  
...  
/dev/tip550_x
```

This pathname must be used in the application program to open a path to the desired TIP550 device.

```
fd = open("/dev/tip550_0", O_RDWR);
```

For debugging you can start the TIP550 Resource Manager with the *-v* option. Now the Resource Manager will print versatile information about TIP550 configuration and command execution on the terminal window.

```
tip550 -v -A<IOaddress> &
```

2.5 Configure module

After starting the driver, the module configuration must be set. You must call the `devctl()`-function as described in chapter 3.3.3 DCMD_T550_VRGCONFIG.

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 TIP550 named by *pathname*.

The flags argument controls how the file is to be opened. TIP550 devices must be opened *O_RDWR*.

EXAMPLE

```
int    fd;

fd = open("/dev/tip550_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 TIP550 driver and should be set to NULL.

The following devctl command codes are defined in *TIP550.h* :

Value	Meaning
<i>DCMD_T550_WRITE</i>	Write DAC Output Value
<i>DCMD_T550_PARAM</i>	Read Module Parameters
<i>DCMD_T550_VRGCONFIG</i>	Configure Voltage Ranges

See behind for more detailed information on each control code.

Note

To use these TIP550 specific control codes the header file
TIP550.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 devctl code separately. Note, the TIP550 driver always returns standard QNX error codes.

SEE ALSO

Library Reference - devctl()

3.3.1 DCMD_T550_WRITE

NAME

DCMD_T550_WRITE - Write to DAC Channel

DESCRIPTION

This devctl function writes a specified value to the specified output channel. A pointer to the callers write buffer (*T550_WRITE_BUF*) and the size of this structure is passed by the parameters *data_ptr* and *n_bytes* to the device.

The *T550_WRITE_BUF* structure has the following layout:

```
typedef struct
{
    /* INPUT: */
    unsigned long    channel;          /* channel number: Sngl:1-16 or Diff:1-8 */
    unsigned long    flags;            /* TIP550_FL_CORR */
    int              data;             /* new DAC value */
} T550_WRITE_BUF, *PT550_WRITE_BUF;
```

channel

Specifies the DAC output channel. The channel number must be between 1 and 8 for TIP550-10 and between 1 and 4 for TIP550-11.

flags

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

TIP550_FL_CORR If set data correction is enabled

data

This value specifies the new output value. The value must be between -2048 and 2047, if voltage range is configured to -10V..+10V and the value must be between 0 and 4096 if the voltage range is configured to 0V..10V.

EXAMPLE

```
int          fd;
int          result;
T550_WRITE_BUF WriteBuf;

...

/* Write 0x123 to DAC channel 1, with corrected data */
WriteBuf.channel = 1;
WriteBuf.flags   = TIP550_FL_CORR;
WriteBuf.data    = 0x123;
result = devctl (fd,
                DCMD_T550_WRITE,
                &WriteBuf,
                sizeof(WriteBuf),
                NULL);
if (result == EOK)
{
    /* DAC channel write 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_T550_PARAM

NAME

DCMD_T550_PARAM - Read from module parameter

DESCRIPTION

This devctl function reads the actual value the specified input channel. A pointer to the callers parameter buffer (*T550_PARAM_BUF*) and the size of this structure is passed by the parameters *data_ptr* and *n_bytes* to the device.

The *T550_PARAM_BUF* structure has the following layout:

```
typedef struct
{
    /* OUTPUT: */
    unsigned long    modeltype;           /* returns modeltype (TIP550-xx) */
    unsigned long    v_range1;           /* voltage selection channel 1...4 */
    unsigned long    v_range2;           /* voltage selection channel 5...8 */
    int              DAC_offset_corr[8]; /* DAC offset correction data */
    int              DAC_gain_corr[8];   /* DAC gain correction data */
} T550_PARAM_BUF, *PT550_PARAM_BUF;
```

modeltype

This argument returns the model type of the TIP550.

v_range1

v_range2

This arguments return the voltage selection of channel 1..4 (*v_range1*) and 5..8 (*v_range2*). The table below shows the possible values.

value	description
TIP550_FL_0_10	The channels are configured for unipolar mode. (0V .. 10V)
TIP550_FL_10_10	The channels are configured for bipolar mode. (-10V .. +10V)

DAC_offset_corr[]

DAC_gain_corr[]

These arrays return the correction data of the TIP550 for output. These values will be used by the driver if data correction is enabled for the write function.

EXAMPLE

```
int          fd;
int          result;
T550_PARAM_BUF ParamBuf;

...

/* Read module parameters */
result = devctl (fd,
                DCMD_T550_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()

3.3.3 DCMD_T550_VRGCONFIG

NAME

DCMD_T550_VRGCONFIG - Configure voltage ranges

DESCRIPTION

This devctl function configures the output ranges of the module. The specified configuration must match to the hardware configuration, otherwise there will be a strange behavior. A pointer to the callers write buffer (*T550_CONFIG_BUF*) and the size of this structure is passed by the parameters *data_ptr* and *n_bytes* to the device.

The *T550_CONFIG_BUF* structure has the following layout:

```
typedef struct
{
    /* INPUT: */
    unsigned long    v_range1;      /* voltage range group 1 (#1..#4)
                                     [TIP550_FL_0_10 | TIP550_FL_10_10] */
    unsigned long    v_range2;      /* voltage range group 2 (#5..#8)
                                     [TIP550_FL_0_10 | TIP550_FL_10_10] */
} T550_CONFIG_BUF, *PT550_CONFIG_BUF;
```

v_range1

v_range2

Specify the DAC output ranges. *v_range1* specifies the output range of channel 1..4 and *v_range2* specifies the output range of channel 5..8. The table below shows the allowed values.

value	description
TIP550_FL_0_10	The channels are configured for unipolar mode. (0V .. 10V)
TIP550_FL_10_10	The channels are configured for bipolar mode. (-10V .. +10V)

EXAMPLE

```
int          fd;
int          result;
T550_CONFIG_BUF ConfigBuf;

...

/* Set channel 1..4: 0..10V, channel 5..8: +/-10V */
ConfigBuf.v_range1 = TIP550_FL_0_10;
ConfigBuf.v_range2 = TIP550_FL_10_10;
result = devctl (fd,
                 DCMD_T550_VRGCONFIG,
                 &ConfigBuf,
                 sizeof(ConfigBuf),
                 NULL);
if (result == EOK)
{
    /* Range configuration write 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()