The Embedded I/O Company

TEWS TECHNOLOGIES

TIP551-SW-95

QNX-Neutrino Device Driver

Optically Isolated 4 Channel 16-Bit DAC

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

TIP551-SW-95

QNX-Neutrino Device Driver

Optically Isolated 4 Channel 16-Bit DAC

Supported Modules:
TIP551

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

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

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

©2004-2009 by TEWS TECHNOLOGIES GmbH

Issue	Description	Date
1.0	First Issue	October 13, 2004
1.1.0	Revision	October 21, 2009

Table of Contents

1 Introduction

The TIP551-SW-95 QNX-Neutrino device driver allows the operation of a TIP551 Optically Isolated 4 Channel 16-Bit DAC IP on QNX-Neutrino operating systems.

The TIP551 device driver is basically implemented as a user installable Resource Manager and started by the TEWS IPAC Carrier Driver (CARRIER-SW-95) if a TIP551 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 TIP551-SW-95 device driver supports the following features:

- Writing new values to a specified DAC channel
- Writing new values to multiple DAC channels with simultaneous update
- Reading device information data
- Automatic offset and gain correction with factory calibration data

The TIP551-SW-95 device driver supports the modules listed below:

TIP551-10	Optically isolated 4 Channel 16 bit D/A, 0V to +10V or +/-10V Output Voltage Range	(IndustryPack®)

To get more information about the features and use of TIP551 devices it is recommended to read the manuals listed below.

TIP551 User manual

TIP551 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 TIP551-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 'TIP551-SW-95':

TIP551-SW-95-SRC.tar.gz	GZIP compressed archive with driver source code
TIP551-SW-95-1.1.0.pdf	PDF copy of this manual
ChangeLog.txt	Release history
Release.txt	Release information

The GZIP compressed archive TIP551-SW-95-SRC.tar.gz contains the following files and directories:

Directory path 'tip551':

driver/tip551.c	Device driver source
driver/tip551.h	Device driver and application include file
driver/tip551def.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/tip551exa.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 TIP551-SW-95-SRC.tar.gz to /usr/src and extract all files (e.g tar -xzvf TIP551-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 tip551.

Change to the driver directory */usr/src/tip551/driver* and copy the header file *tip551.h* to */usr/include* allowing user application programs sharing the TIP551 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 TIP551-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/tip551/driver* and execute the Makefile

```
# make install
```

After successful completion the driver dynamic library *tip551.so* will be installed in the directory */lib/dll*.

2.2 Build the example application

Change to the directory */usr/src/tip551/example* and execute the Makefile

```
# make install
```

After successful completion the example binary *tip551exa* will be installed in the directory */bin*.

2.3 Start the driver process

To start the TIP551 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 TIP551 resource manager registers a device for each TIP551 in the QNX-Neutrinos pathname space. The device file /dev/tip551_0 belongs to the first TIP551 found, the device file /dev/tip551_1 to the second TIP551 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 TIP551 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 TIP551 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 TIP551 named by *pathname.* The flags argument controls how the file is to be opened. TIP551 devices must be opened *O_RDWR*.

EXAMPLE

```
int  fd;

fd = open("/dev/tip551_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 TIP551 driver and should be set to NULL.

The following devctl command codes are defined in *tip551.h* :

Value	**Meaning**
DCMD_TIP551_WRITE	Write new data to specified DAC channel(s)
DCMD_TIP551_GET_INFO	Get device information

See behind for more detailed information on each control code.

To use these TIP551 specific control codes the header file tip551.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 TIP551 driver always returns standard QNX Neutrino error codes.

SEE ALSO

Library Reference - devctl()

3.3.1 DCMD_TIP551_WRITE

NAME

DCMD_TIP551_WRITE – Write new data to the specified DAC channel(s)

DESCRIPTION

This function attempts to write new data to the specified DAC channel(s) of the TIP551 associated with the file descriptor filedes from a structure (*T551_WRITE_BUFFER*) pointed by data_ptr. The argument n_bytes specifies the length of the write buffer.

```
typedef struct
{
        int     enable[T551_MAX_CHAN];
        int     corr[T551_MAX_CHAN];
        int     data[T551_MAX_CHAN];
} T551_WRITE_BUFFER;
```

The write function supports the simultaneous update feature of the TIP551. It's possible to update the DAC outputs of up to 4 channels with one write. In this case the internal DAC data registers of the specified DAC channels will be loaded with new data and then the data conversion is started simultaneously at all DAC channels.

enable

The field enable[0] controls DA conversion of TIP551 DAC Channel 1, enable[1] for Channel 2 and so forth. If enable[n] is TRUE (1), the associated channel is going to be updated. Setting enable[n] equal to FALSE (0) disables DA conversion of the associated channel. If more than one channel is enabled, simultaneous update feature is automatically chosen.

corr

Set this parameter to TRUE (1) to perform an automatic gain and offset correction with factory calibration data stored in the IDPROM. FALSE (0) means do not perform any correction and write directly to the DAC output.

data

Contains the new DAC output value. For further information about unipolar or bipolar modes, see TIP551 hardware manual.

Output Mode	Data Range	Voltage Range
Unipolar	0...65535	0...10V
Bipolar	-32768...32767	-10V...10V

EXAMPLE

```
#include <tip551.h>

int                 fd;
int                 result;
T551_WRITE_BUFFER   wrBuf;
/*
** Update only Channel 1
** pre-cond.: 1. wrBuf is filled with 0x00
**            2. fd is a valid device handle
*/
wrBuf.enable[T551_CH1]  = TRUE;        /* enable updating of channel 1 */
wrBuf.corr[T551_CH1]    = TRUE;        /* data correction on */
wrBuf.data[T551_CH1]    = 65535;       /* full-scale if unipolar output */

result = devctl(fd, DCMD_TIP551_WRITE, &wrBuf, sizeof(wrBuf), NULL);
if (result == EOK)
{
     printf("\nDCMD_TIP551_WRITE successful\n");
}
else { /* report failure */ }

/*
** Update channel 1, 2, 3 with new data and let channel 4
** unchanged.
*/
wrBuf.enable[T551_CH3]  = TRUE;        /* enable updating of channel 3 */
wrBuf.corr[T551_CH3]    = TRUE;        /* data correction */
wrBuf.data[T551_CH3]    = -32768;      /* -full-scale if bipolar output */

wrBuf.enable[T551_CH1]  = TRUE;        /* enable updating of channel 1 */
wrBuf.corr[T551_CH1]    = TRUE;        /* data correction */
wrBuf.data[T551_CH1]    = 0;           /* mid-scale if bipolar output */

wrBuf.enable[T551_CH2]  = TRUE;        /* enable updating of channel 2 */
wrBuf.corr[T551_CH2]    = TRUE;        /* data correction */
wrBuf.data[T551_CH2]    = 32767;       /* +full-scale if bipolar output */

result = devctl(fd, DCMD_TIP551_WRITE, &wrBuf, sizeof(wrBuf), NULL);

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

RETURNS

On success DCMD_TIP551_WRITE returns EOK. In the case of an error, the appropriate error code is returned by the function (not in errno!).

ERRORS

EFAULT	Invalid buffer pointer. Please check the argument data_ptr.
EINVAL	The size of the buffer is too small
ETIME	Timeout during DA conversion.

SEE ALSO

Library Reference - devctl()

3.3.2 DCMD_TIP551_GET_INFO

NAME

DCMD_TIP551_GET_INFO - Get device information

DESCRIPTION

This ioctl function reads the module variant, the current output voltage range and the factory calibration data from the specified device and returns this information in the *T551_INFO_BUFFER* structure to the caller.

A pointer to the *T551_INFO_BUFFER* structure is passed by the parameter *data_ptr* to the driver.

```
typedef struct
{
        int     variant;
        int     voltage_range;
        short   offset_corr[T551_MAX_CHAN];
        short   gain_corr[T551_MAX_CHAN];
} T551_INFO_BUFFER;
```

variant

This parameter returns the module variant.

Value	**Module Variant**
10	TIP551-10

voltage_range

Returns the actual output voltage range for all channels.

Vaue	**Symbol**	**Voltage Range**
1	T551_UNIPOLAR	0V ... 10V
2	T551_BIPOLAR	-10V ... 10V

offset_corr

Returns the factory offset calibration data for channel 1...4 in the unit ¼ LSB.

gain_corr

Returns the factory gain calibration data for channel 1...4 in the unit ¼ LSB.

EXAMPLE

```
#include <tip551.h>

int                 fd;
int                 result;
T551_INFO_BUFFER    infoBuf;

result = devctl(fd, DCMD_TIP551_GET_INFO, &infoBuf, sizeof(infoBuf), NULL);

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

ERRORS

EFAULT	Invalid buffer pointer. Please check the argument data_ptr.
EINVAL	The size of the buffer is too small

SEE ALSO

Library Reference - devctl()