The Embedded I/O Company

TEWS TECHNOLOGIES

TIP710-SW-95

QNX-Neutrino Device Driver

16 Digital Outputs

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

TIP710-SW-95

QNX-Neutrino Device Driver

16 Digital Outputs

Supported Modules:
TIP710

Issue	Description	Date
1.0	First Issue	October 13, 2004
1.1.0	Revision	October 22, 2009

Table of Contents

1 Introduction

The TIP710-SW-95 QNX-Neutrino device driver allows the operation of a TIP710 16 Digital Outputs IP on QNX-Neutrino operating systems.

The TIP710 device driver is basically implemented as a user installable Resource Manager and started by the TEWS IPAC Carrier Driver (CARRIER-SW-95) if a TIP710 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 TIP710-SW-95 device driver supports the following features:

- Writing digital output value
- Enable/disable output watchdog

The TIP710-SW-95 device driver supports the modules listed below:

TIP710-10	16 Digital Outputs, 6V to 48V DC High Side Switch	(IndustryPack®)

To get more information about the features and use of TIP710 devices it is recommended to read the manuals listed below.

TIP710 User manual

TIP710 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 TIP710-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 'TIP710-SW-95':

TIP710-SW-95-SRC.tar.gz	GZIP compressed archive with driver source code
TIP710-SW-95-1.1.0.pdf	PDF copy of this manual
ChangeLog.txt	Release history
Release.txt	Release information

The GZIP compressed archive TIP710-SW-95-SRC.tar.gz contains the following files and directories:

Directory path 'tip710':

driver/tip710.c	Device driver source
driver/tip710.h	Device driver and application include file
driver/tip710def.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/tip710exa.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 TIP710-SW-95-SRC.tar.gz to /usr/src and extract all files (e.g tar -xzvf TIP710-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 tip710.

Change to the driver directory */usr/src/tip710/driver* and copy the header file *tip710.h* to */usr/include* allowing user application programs sharing the TIP710 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 TIP710-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/tip710/driver* and execute the Makefile

```
# make install
```

After successful completion the driver dynamic library *tip710.so* will be installed in the directory */lib/dll*.

2.2 Build the example application

Change to the directory */usr/src/tip710/example* and execute the Makefile

```
# make install
```

After successful completion the example binary *tip710exa* will be installed in the directory */bin*.

2.3 Start the driver process

To start the TIP710 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 TIP710 resource manager registers a device for each TIP710 in the QNX-Neutrinos pathname space. The device file /dev/tip710_0 belongs to the first TIP710 found, the device file /dev/tip710_1 to the second TIP710 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 TIP710 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 TIP710 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 TIP710 named by *pathname.* The flags argument controls how the file is to be opened. TIP710 devices must be opened *O_RDWR*.

EXAMPLE

```
int  fd;

fd = open(“/dev/tip710_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 TIP710 driver and should be set to NULL.

The following devctl command codes are defined in *tip710.h* :

Value	Meaning
DCMD_TIP710_WRITE	Write digital output value
DCMD_TIP710_ENABLE_WD	Enable the output watchdog
DCMD_TIP710_DISABLE_WD	Disable the output watchdog

See behind for more detailed information on each control code.

To use these TIP710 specific control codes the header file tip710.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 TIP710 driver always returns standard QNX Neutrino error codes.

SEE ALSO

Library Reference - devctl()

3.3.1 DCMD_TIP710_WRITE

NAME

DCMD_TIP710_WRITE – Write digital output value

DESCRIPTION

This function attempts to write to the output registers of the TIP710 associated with the file descriptor *filedes.* A pointer to an unsigned short variable is passed by *data_ptr*.to the driver. The argument size should always be sizeof(unsigned short) and is passed by *n_bytes*.

Bit 0 of the unsigned short value corresponds to output line 1, Bit 1 corresponds to output line 2 and so on.

EXAMPLE

```
#include <tip710.h>

unsigned short outval = 0xF1A3;

/*
**  Send request to the device driver
*/
result = devctl(fd, DCMD_TIP710_WRITE, &outVal, sizeof(outVal), NULL);

/*
**  Check the result of the last device I/O operation
*/
if( result == EOK)
{
     printf("\nWrite value successful\n");
}
else
{
     printf("devctl failed (Error = %d) : %s\n", result, strerror(result));
}
```

RETURNS

On success DCMD_TIP710_WRITE returns EOK. In the case of an error, the appropriate error code is returned by the function (not in errno!).

3.3.2 DCMD_TIP710_ENABLE_WD

NAME

DCMD_TIP710_ENABLE_WD - Enable output watchdog

DESCRIPTION

This function enables the output watchdog facility of the TIP710. If the output watchdog is not retriggered within approximately 120 milliseconds the state of the output register will be set to inactive. The watchdog is triggered implicitly by a write to the output register.

The initialization state (driver startup) of the output watchdog is disabled.

EXAMPLE

```
#include <tip710.h>

/*
**  Send request to the device driver
*/
result = devctl(fd, DCMD_TIP710_ENABLE_WD, NULL, 0, NULL);

/*
**  Check the result of the last device I/O operation
*/
if( result == EOK)
{
     printf("\nEnable watchdog successful\n");
}
else
{
     printf("devctl failed (Error = %d) : %s\n", result, strerror(result));
}
```

3.3.3 DCMD_TIP710_DISABLE_WD

NAME

DCMD_TIP710_DISABLE_WD - Disable output watchdog

DESCRIPTION

This function disables the output watchdog facility of the TIP710.

The initialization state (driver startup) of the output watchdog is disabled.

EXAMPLE

```
#include <tip710.h>

/*
**  Send request to the device driver
*/
result = devctl(fd, DCMD_TIP710_DISABLE_WD, NULL, 0, NULL);

/*
**  Check the result of the last device I/O operation
*/
if( result == EOK)
{
     printf("\nDisable watchdog successful\n");
}
else
{
     printf("devctl failed (Error = %d) : %s\n", result, strerror(result));
}
```