The Embedded I/O Company

TEWS TECHNOLOGIES

TIP710-SW-82

Linux Device Driver

16 Digital Outputs, 6V to 48V DC
High Side Switch

Version 1.2.x

User Manual

Issue 1.2.2

February 2009

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

TEWS TECHNOLOGIES LLC
9190 Double Diamond Parkway,
Suite 127, Reno, NV 89521, USA
www.tews.com
Phone: +1 (775) 850 5830
Fax: +1 (775) 201 0347
e-mail: usasales@tews.com

TIP710-SW-82

Linux Device Driver

16 Digital Outputs, 6V to 48V DC
High Side Switch

Supported Modules:
TIP710

Issue	Description	Date
1.0	First Issue	June 17, 2003
1.1	Support for IPAC CARRIER DRIVER, DEVFS and SMP	February 10, 2004
1.2.0	Introduction and installation section modified	June 8, 2006
1.2.1	New Address TEWS LLC, ChangeLog.txt added to file list Extract information modified	June 22, 2007
1.2.2	New description for IPAC Carrier Driver	February 25, 2009

Table of Contents

1 Introduction

The TIP710-SW-82 Linux device driver allows the operation of TIP710 IPAC modules on Linux operating systems.

Because the TIP710 device driver is stacked on the TEWS TECHNOLOGIES IPAC carrier driver, it's necessary to install also the appropriate IPAC carrier driver. Please refer to the IPAC carrier driver user manual for further information.

The TIP710 device driver includes the following features:

- writing digital output value
- enable and disable output watchdog

The TIP710-SW-82 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 the supported devices it is recommended to read the manuals listed below.

TIP710 User manual

TIP710 Engineering Manual

CARRIER-SW-82 IPAC Carrier 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 so on. Also the different byte ordering (big-endian versus little-endian) of CPU boards will cause problems on accessing the IndustryPack I/O and memory spaces.

To simplify the implementation of IPAC device drivers which work with any 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-82 is part of this TIP710-SW-82 distribution. It is located in directory CARRIER-SW-82 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-82 User Manual for a detailed description how to install and setup the CARRIER-SW-82 device driver, and for a description of the TEWS TECHNOLOGIES IPAC Carrier Driver concept.

2 Installation

The directory TIP710-SW-82 on the distribution media contains the following files:

TIP710-SW-82-1.2.2.pdf	This manual in PDF format
TIP710-SW-82-SRC.tar.gz	GZIP compressed archive with driver source code
Release.txt	Release information
ChangeLog.txt	Release history

The GZIP compressed archive TIP710-SW-82-SRC.tar.gz contains the following files and directories:

Directory path './tip710/':

tip710.c	Driver source code
tip710def.h	Driver include file
tip710.h	Driver include file for application program
makenode	Script to create device nodes on the file system
Makefile	Device driver make file
example/tip710exa.c	Example application
example/Makefile	Example application make file
include/config.h	Driver independent library header file
include/tpmodule.h	Kernel independent library header file
include/tpmodule.c	Kernel independent library source code file

In order to perform an installation, extract all files of the archive TIP710-SW-82-SRC.tar.gz to the desired target directory. The command 'tar -xzvf TIP710-SW-82-SRC.tar.gz' will extract the files into the local directory.

Before building a new device driver, the TEWS TECHNOLOGIES IPAC carrier driver must be installed properly, because this driver includes the header file *ipac_carrier.h*, which is part of the IPAC carrier driver distribution. Please refer to the IPAC carrier driver user manual in the directory path *CARRIER-SW-82* on the separate distribution media.

2.1 Build and install the device driver

- Login as *root*
- Change to the target directory
- To create and install the driver in the module directory */lib/modules/<version>/misc* enter:

  # make install

  For Linux kernel 2.6.x, there may be compiler warnings claiming some undefined ipac_* symbols. These warnings are caused by the IPAC carrier driver, which is unknown during compilation of this TIP driver. The warnings can be ignored.

- Also after the first build we have to execute *depmod* to create a new dependency description for loadable kernel modules. This dependency file is later used by *modprobe* to automatically load the correct IPAC carrier driver modules.

  # depmod –aq

2.2 Uninstall the device driver

- Login as *root*
- Change to the target directory
- To remove the driver from the module directory */lib/modules/<version>/misc* enter:

  # make uninstall

- Update kernel module dependency description file:

  # depmod –aq

2.3 Install device driver into the running kernel

- To load the device driver into the running kernel, login as root and execute the following commands:

  # modprobe tip710drv

- After the first build or if you are using dynamic major device allocation it's necessary to create new device nodes on the file system. Please execute the script file *makenode* to do this. If your kernel has enabled a dynamic device file system (devfs or sysfs with udev) then you have to skip running the *makenode* script. Instead of creating device nodes from the script the driver itself takes creating and destroying of device nodes in its responsibility.

  # sh makenode

On success the device driver will create a minor device for each TIP710 module found. The first TIP710 can be accessed with device node /dev/tip710_0, the second TIP710 with device node /dev/tip710_1, the third TIP710 with device node /dev/tip710_2 and so on.

The allocation of device nodes to physical TIP710 modules depends on the search order of the IPAC carrier driver. Please refer to the IPAC carrier user manual.

Loading of the TIP710 device driver will only work if kernel KMOD support is installed, necessary carrier board drivers already installed and the kernel dependency file is up to date. If KMOD support isn't available you have to build either a new kernel with KMOD installed or you have to install the IPAC carrier kernel modules manually in the correct order (please refer to the IPAC carrier driver user manual).

2.4 Remove device driver from the running kernel

- To remove the device driver from the running kernel login as root and execute the following command:

  # modprobe tip710drv –r

If your kernel has enabled a dynamic device file system (devfs or sysfs with udev) all /dev/tip710_x nodes will be automatically removed from your file system after this.

Be sure that the driver isn't opened by any application program. If opened you will get the response "*tip710drv: Device or resource busy*" and the driver will still remain in the system until you close all opened files and execute *modprobe –r* again.

2.5 Change Major Device Number

The TIP710 driver uses dynamic allocation of major device numbers by default. If this isn't suitable for the application it's possible to define a major number for the driver. If the kernel has enabled a dynamic files system the driver will not use the symbol TIP710_MAJOR.

To change the major number edit the file tip710.c, change the following symbol to appropriate value and enter **make install** to create a new driver.

TIP710_MAJOR	Valid numbers are in range between 0 and 255. A value of 0 means dynamic number allocation.

Example:

```
#define TIP710_MAJOR       122
```

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 <fcntl.h>

int open (const char *filename, int flags)

DESCRIPTION

The open function creates and returns a new file descriptor for the file named by *filename*. The *flags* argument controls how the file is to be opened. This is a bit mask; you create the value by the bitwise OR of the appropriate parameters (using the | operator in C). See also the GNU C Library documentation for more information about the open function and open flags.

EXAMPLE

```
int fd;

fd = open(“/dev/tip710_0”, O_RDWR);
if (fd < 0)
{
   /* handle error condition */
}
```

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

E_NODEV	The requested minor device does not exist.

This is the only error code returned by the driver, other codes may be returned by the I/O system during open.

SEE ALSO

GNU C Library description – Low-Level Input/Output

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

E_NODEV	The requested minor device does not exist.

This is the only error code returned by the driver, other codes may be returned by the I/O system during close. For more information about close error codes, see the *GNU C Library description – Low-Level Input/Output*.

SEE ALSO

GNU C Library description – Low-Level Input/Output

3.3 write()

NAME

write() – write to a device

SYNOPSIS

#include <unistd.h>

ssize_t write(int filedes, void *buffer, size_t size)

DESCRIPTION

This function attempts to write to the output registers of the TIP710 associated with the file descriptor *filedes* from a structure (*T710_BUFFER)* pointed to by *buffer*. The argument *size* specifies the length of the buffer and must be set to the size of the structure *T710_BUFFER*.

typedef struct
{
    unsigned short   value;
} T710_BUFFER;

value

This parameter holds the new value for output lines 1 to 16. Where bit 2^0 corresponds to output line 1, bit 2^1 to output line 2, and so on.

EXAMPLE

```
#include “tip710.h”

int         fd;
ssize_t     NumBytes;
T710_BUFFER ioBuf;

ioBuf.value = 0x1234;

NumBytes = write(fd, &ioBuf, sizeof(T710_BUFFER));

if (NumBytes != sizeof(T710_BUFFER)) {
   // process error;
}
```

RETURNS

On success write returns the size of bytes written (always the size of *T710_BUFFER*). In the case of an error, a value of –1 is returned. The global variable *errno* contains the detailed error code.

ERRORS

EINVAL	This error code is returned if the size of the buffer is wrong.

SEE ALSO

GNU C Library description – Low-Level Input/Output

3.4 ioctl()

NAME

ioctl() – device control functions

SYNOPSIS

#include <sys/ioctl.h>

int ioctl(int filedes, int request [, void *argp])

DESCRIPTION

The **ioctl** function sends a control code directly to a device, specified by *filedes*, causing the corresponding device to perform the requested operation.

The argument *request* specifies the control code for the operation. The optional argument *argp* depends on the selected request and is described for each request in detail later in this chapter.

The following ioctl codes are defined in *tip710.h*:

Symbol	Meaning
T710_IOC_ENABLE_WD	Enable output watchdog
T710_IOC_DISABLE_WD	Disable output watchdog
T710_IOC_TRIGGER_WD	Trigger 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, zero is returned. In the case of an error, a value of –1 is returned. The global variable *errno* contains the detailed error code.

ERRORS

EINVAL	Invalid argument. This error code is returned if the requested ioctl function is unknown. Please check the argument *request*.

Other function dependant error codes will be described for each ioctl code separately. Note, the TIP710 driver always returns standard Linux error codes.

SEE ALSO

ioctl man pages

3.4.1 T710_IOC_ENABLE_WD

NAME

T710_IOC_ENABLE_WD - Enable output watchdog

DESCRIPTION

This ioctl 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 or by executing the ioctl function *T710_IOC_TRIGGER_WD*.

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

EXAMPLE

```
#include “tip710.h”

int  fd;
int  result;

result = ioctl(fd, T710_IOC_ENABLE_WD);

if (result < 0) {
   /* handle ioctl error */
}
```

ERRORS

This ioctl function returns no function specific error codes.

3.4.2 T710_IOC_DISABLE_WD

NAME

T710_IOC_DISABLE_WD - Disable output watchdog

DESCRIPTION

This ioctl function disables the output watchdog facility of the TIP710.

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

EXAMPLE

```
#include “tip710.h”

int  fd;
int  result;

result = ioctl(fd, T710_IOC_DISABLE_WD);

if (result < 0) {
   /* handle ioctl error */
}
```

ERRORS

This ioctl function returns no function specific error codes.

3.4.3 T710_IOC_TRIGGER_WD

NAME

T710_IOC_TRIGGER_WD - Trigger output watchdog

DESCRIPTION

This ioctl function triggers the output watchdog 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 or by executing this ioctl function.

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

EXAMPLE

```
#include “tip710.h”

int  fd;
int  result;

result = ioctl(fd, T710_IOC_TRIGGER_WD);

if (result < 0) {
   /* handle ioctl error */
}
```

ERRORS

This ioctl function returns no function specific error codes.

4 Debugging

For debugging output see tip710.c. You will find the following symbol:

```
#undef TIP710_DEBUG_VIEW
```

To enable a debug output replace “undef” with “define”.

The TIP710_DEBUG_VIEW symbol controls debugging output from the whole driver.

You can retrieve these messages from the /proc file system using the following command:

 # cat /proc/kmsg

```
TIP710  :  Probe new TIP710 mounted on <TEWS TECHNOLOGIES - (Compact)PCI
IPAC Carrier> at slot B

TIP710  :  IP I/O Memory Space
00000000 : 00 00 FF 00

TIP710  :  IP ID Memory Space
00000000 : FF 49 FF 50 FF 41 FF 43 FF B3 FF 33 FF 10 FF 00
00000010 : FF 00 FF 00 FF 0C FF 7F FF 00 FF 00 FF 00 FF 00
00000020 : FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00
00000030 : FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00
```