The Embedded I/O Company

TEWS TECHNOLOGIES

TIP675-SW-82

Linux Device Driver

48 TTL I/O Lines with Interrupts

Version 1.2.x

User Manual

Issue 1.2.3

July 2008

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

TIP675-SW-82

Linux Device Driver

48 TTL I/O Lines with Interrupts

Supported Modules:
TIP675

Issue	Description	Date
1.0	First Issue	December 11, 2002
1.1	Support for DEVFS and SMP	February 17, 2004
1.2.0	Kernel 2.6.x Support	January 13, 2006
1.2.1	New Address TEWS LLC	September 28, 2006
1.2.2	General revision	February 13, 2008
1.2.3	Carrier Driver description added	July 7, 2008

Table of Contents

1 Introduction

1.1 Device Driver

The TIP675-SW-82 Linux device driver allows the operation of a TIP675 IPAC module on Linux operating systems.

Because the TIP675 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 TIP675-SW-82 device driver includes the following features:

- reading the input register
- writing the output register
- programming direction of every I/O line
- configure simultaneous update feature
- waiting for a transition at a single input line or a group of input lines (OR'ed)
- designed as Linux kernel module with dynamically loading (modprobe).
- creates devices with dynamically allocated or fixed major device numbers.
- Support for dynamic device file systems
- TEWS TECHNOLOGIES IPAC carrier driver support

The TIP675-SW-82 supports the modules listed below:

TIP675-10	48 TTL I/O Lines with Interrupts	(IndustryPack ®)

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

TIP675 User Manual

TIP675 Engineering Manual

CARRIER-SW-82 IPAC Carrier User Manual

1.2 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 TIP675-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 TIP675-SW-82 on the distribution media contains the following files:

TIP675-SW-82-1.2.3.pdf	This manual in PDF format
TIP675-SW-82-SRC.tar.gz	GZIP compressed archive with driver source code
ChangeLog.txt	Release history
Release.txt	Release information

The GZIP compressed archive TIP675-SW-82-SRC.tar.gz contains the following files and directories:

All paths are relative to subdirectory TIP675:

tip675.c	Driver source code
tip675def.h	Driver include file
tip675.h	Driver include file for application program
makenode	Script to create device nodes on the file system
Makefile	Device driver make file
example/tip675exa.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 TIP675-SW-82-SRC.tar.gz to the desired target directory. The command 'tar -xzvf TIP675-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 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 tip675drv

- After the first build or if you are using dynamic major device allocation it is 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 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 TIP675 module found. The first TIP675 can be accessed with device node /dev/tip675_0, the second TIP675 or the second channel of the first TIP675 with device node /dev/tip675_1 and so on.

The allocation of device nodes to physical TIP675 modules depends on the search order of the IPAC carrier driver. Please refer to the IPAC carrier user manual.

Loading of the TIP675 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 tip675drv -r

If your kernel has enabled a dynamic device file system all /dev/tip675_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 *´´tip675drv: 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 TIP675 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 devfs the driver will not use the symbol TIP675_MAJOR.

To change the major number edit the file tip675.c, change the following symbol to appropriate value and enter **make install** to create a new driver.

TIP675_MAJOR	Valid numbers are in range between 0 and 255. A value of 0 means dynamic number allocation.

Example:

```
#define TIP675_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/tip675_0”, O_RDWR);
if (fd == -1)
{
  /* 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

ENODEV	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

ENODEV	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 read()

NAME

read() – read from a device

SYNOPSIS

#include <unistd.h>

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

DESCRIPTION

This function attempts to read the input registers of the TIP675 associated with the file descriptor *fildes* into a structure (*T675_BUFFER*) pointed by *buffer*. The argument *size* specifies the length of the buffer and must be set to the length of the structure *T675_BUFFER*.

```
typedef struct
{
        unsigned short      line_1_16;
        unsigned short      line_17_32;
        unsigned short      line_33_48;
} T675_BUFFER;
```

line_1_16

This parameter returns the status of input lines 1 to 16. Where bit 2^0 corresponds to input line 1, bit 2^1 to input line 2, and so on.

line_17_32

This parameter returns the status of input lines 17 to 32. Where bit 2^0 corresponds to input line 17, bit 2^1 to input line 18, and so on.

line_33_48

This parameter returns the status of input lines 33 to 48. Where bit 2^0 corresponds to input line 33, bit 2^1 to input line 34, and so on.

EXAMPLE

```
#include <tip675.h>

int  fd;
ssize_t  num_bytes;
T675_BUFFER  io_buf;

num_bytes = read(fd, &io_buf, sizeof(io_buf));

if (num_bytes != sizeof(T675_BUFFER))
{
   // process error;
}
```

RETURNS

On success read returns the number of byte read (always size of *T675_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 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 TIP675 associated with the file descriptor *filedes* from a structure (*T675_BUFFER)* pointed by *buffer*. The argument *size* specifies the length of the buffer and must be set to the length of the structure *T675_BUFFER*.

```
typedef struct
{
        unsigned short      line_1_16;
        unsigned short      line_17_32;
        unsigned short      line_33_48;
} T675_BUFFER;
```

line_1_16

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.

line_17_32

This parameter holds the new value for output lines 17 to 32. Where bit 2^0 corresponds to output line 17, bit 2^1 to output line 18, and so on.

line_33_48

This parameter holds the new value for output lines 33 to 48. Where bit 2^0 corresponds to output line 33, bit 2^1 to output line 34, and so on.

EXAMPLE

```
#include <tip675.h>

int fd;
ssize_t  num_bytes;
T675_BUFFER  io_buf;

// set OUTPUT 1,16,31 and 48 to logic high
io_buf.line_1_16   = 0x8001;
io_buf.line_17_32  = 0x4000;
io_buf.line_33_48  = 0x8000;

num_bytes = write(fd, &io_buf, sizeof(T675_BUFFER));

if (num_bytes != sizeof(T675_BUFFER))
{
  // process error;
}
```

RETURNS

On success write returns the size of bytes written (always the size of *T675_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

3.5 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 tip*675.h*:

Symbol	Meaning
T675_IOCS_SET_DIR	Set direction of I/O lines
T675_IOCS_ENA_EXCLK	Enable simultaneous update feature
T675_IOC_DIS_EXCLK	Disable simultaneous update feature
T675_IOCX_WAIT_TRANS	Waiting for a transition at a single input line or a group of input lines (OR'ed)

See behind for more detailed information on each control code.

To use these TIP675 specific control codes the header file tip675.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 TIP675 driver always returns standard Linux error codes.

SEE ALSO

ioctl man pages

3.5.1 T675_IOCS_SET_DIR

NAME

T675_IOCS_SET_DIR - Set direction of I/O lines

DESCRIPTION

With this ioctl function each of the 48 I/O lines may be individually set as input or output. To set a line to be input, set the corresponding bit in the mask to 0.

A pointer to the direction mask structure (T675_BUFFER) is passed by the parameter *argp* to the driver.

```
typedef struct
{
        unsigned short      line_1_16;
        unsigned short      line_17_32;
        unsigned short      line_33_48;
} T675_BUFFER;
```

line_1_16

This parameter holds the new direction 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.

line_17_32

This parameter holds the new direction for output lines 17 to 32. Where bit 2^0 corresponds to output line 17, bit 2^1 to output line 18, and so on.

line_33_48

This parameter holds the new direction for output lines 33 to 48. Where bit 2^0 corresponds to output line 33, bit 2^1 to output line 34, and so on.

After driver startup all I/O lines are set to be inputs.

EXAMPLE

```
#include <tip675.h>

int  fd;
int  result;
T675_BUFFER  dir_buf;

// Set line 1...24 to be output and line 25...48 to be input
dir_buf.line_1_16  = 0xFFFF;
dir_buf.line_17_32 = 0x00FF;
dir_buf.line_33_48 = 0x0000;

result = ioctl(fd, T675_IOCS_SET_DIR, &dir_buf);

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

ERRORS

EFAULT	Invalid pointer to the direction mask buffer. Please check the argument argp.

3.5.2 T675_IOCS_ENA_EXCLK

NAME

T675_IOCS_ENA_EXCLK - Enable simultaneous update feature

DESCRIPTION

This ioctl function enables the simultaneous update feature of the TIP675. The argument *argp* passes a pointer to an unsigned long variable to the driver. Setting this variable to a 0 will cause the inputs and outputs to be latched on the rising edge of the external clock, while setting to a 1 will latch the inputs and outputs on the falling edge.

After driver startup the simultaneous update feature is disabled.

EXAMPLE

```
#include <tip675.h>

int  fd;
int  result;
unsigned long  value;

// Enable simultaneous update on falling edge
value = 1;

result = ioctl(fd, T675_IOCS_ENA_EXCLK, &value);

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

ERRORS

EFAULT	Invalid pointer. Please check the argument *argp*.

3.5.3 T675_IOCS_DIS_EXCLK

NAME

T675_IOCS_DIS_EXCLK - Disable simultaneous update feature

DESCRIPTION

This ioctl function disables the simultaneous update feature of the TIP675. The argument *argp* is not needed and can be omitted.

After driver startup the simultaneous update feature is disabled.

EXAMPLE

```
#include <tip675.h>

int  fd;
int  result;

result = ioctl(fd, T675_IOCS_DIS_EXCLK);

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

3.5.4 T675_IOCX_WAIT_TRANS

NAME

T675_IOCX_WAIT_TRANS - Waiting for a specified input line transition

DESCRIPTION

This ioctl function will be blocked until a specified transition on a selected input line has occurred or the maximum allowed time is elapsed. If more than one input line is selected at least one transition must occur (logical OR) to finish this function. On success this function returns the transition status to the caller.

A pointer to the callers parameter buffer (*T675_EVRD_BUFFER*) is passed by the parameter *argp* to the driver.

```
typedef struct
{
        unsigned short      mask_1_16;
        unsigned short      mask_17_32;
        unsigned short      mask_33_48;
        unsigned short      status_1_16;
        unsigned short      status_17_32;
        unsigned short      status_33_48;
        long                mode;
        long                timeout;
} T675_EVRD_BUFFER;
```

mask_1_16, mask_17_32, mask_33_48

These parameters contain a bit mask to select a certain bit position or a group of bits for an input transition detection. Bits 2^0 to 2^{15} of mask_1_16 correspond to Line 1 to 16, Bits 2^0 to 2^{15} of mask_17_32 correspond to Line 17 to 32. A certain input line can be selected by setting the corresponding bit position to 1.

status_1_16, status_17_32, status_33_48

These parameters receive a bit mask of the corresponding input lines, which can be used to determine the source of the event (interrupt). This is useful if more than one bit is selected. If the selected transition has occurred, the corresponding bit position contains a 1.

mode

Specifies the transition mode for this request:

Value	Description
T675_HIGH_TR	In this mode the ioctl function will be blocked until a high transition occurred at the selected input line(s).
T675_LOW_TR	In this mode the ioctl function will be blocked until a low transition occurred at the selected input line(s).
T675_ANY_TR	In this mode the ioctl function will be blocked until a low or high transition occurred at the selected input line(s).

timeout

This parameter specifies the amount of time (in ticks) the caller is willing to wait for the occurrence of the requested transition. A value of 0 means wait indefinitely.

EXAMPLE

```
#include <tip675.h>

int  fd;
int  result;
T675_EVRD_BUFFER  ev_buf;

// Wait for any transition at line 1 or 48
ev_buf.mask_1_16  = 0x0001;
ev_buf.mask_17_32 = 0x0000;
ev_buf.mask_33_48 = 0x8000;
ev_buf.mode       = T675_ANY_TR;
par.timeout       = 1000;

result = ioctl(fd, T675_IOCX_WAIT_TRANS, &ev_buf);

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

ERRORS

EFAULT	Invalid pointer. Please check the argument *argp*.
EBUSY	The maximum number of concurrent read requests was exceeded. Increase the value of *MAX_REQUESTS* in tip675def.h.
ETIME	The allowed time to finish the read request is elapsed.
EINTR	Interrupted function call; an asynchronous signal occurred and prevented completion of the call. When this happens, you should try the call again.

4 Debugging

For debugging output see tip675.c. You will find the two following symbols:

```
#undef TIP675_DEBUG_INTR
#undef TIP675_DEBUG_VIEW
```

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

The TIP675_DEBUG_INTR symbol controls debugging output from the ISR.

```
TIP675  :  interrupt entry
TIP675  :  IACK[0] vector = 0005
```

The TIP675_DEBUG_VIEW symbol controls debugging output from the remaining part of the driver.

```
TIP675 - 48 TTL I/O Lines with Interrupts - version 1.2.2 (2008-02-13)<6>
TIP675  :  Probe new TIP675 mounted on <TEWS TECHNOLOGIES - (Compact)PCI
IPAC Carrier> at slot A

TIP675  : Create minor node /dev/tip675_0 (devfs).

TIP675  :  IP I/O Memory Space
00000000 : FF FF FF FF FF FF FF FF FF FF FF FF 00 00 00 00
00000010 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000020 : 00 00 00 00 00 00 00 00 00 00 00 00 00 05

TIP675  :  IP ID Memory Space
00000000 : 00 49 00 50 00 41 00 43 00 B3 00 36 00 10 00 00
00000010 : 00 00 00 00 00 0C 00 D8 00 00 00 00 00 00 00 00
00000020 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000030 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```