The Embedded I/O Company

TEWS TECHNOLOGIES

TIP672-SW-72

LynxOS Device Driver

24 Differential I/O Lines with Interrupts

Version 2.0.x

User Manual

Issue 2.0

December 2003

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

TEWS TECHNOLOGIES LLC
1 E. Liberty Street, Sixth Floor
Reno, Nevada 89504 / USA
Phone: +1 (775) 686 6077
Fax: +1 (775) 686 6024
e-mail: usasales@tews.com
www.tews.com

TIP672-SW-72

24 Differential I/O Lines with Interrupts

LynxOS Device Driver

Issue	Description	Date
1.0	First Issue	March 31, 2003
2.0	IPAC Carrier Driver Model	December 15, 2003

Table of Contents

1 Introduction

The TIP672-SW-72 LynxOS device driver allows the operation of a TIP672 IPAC module on LynxOS operating systems.

Because the TIP672 device driver is stacked on the TEWS TECHNOLOGIES IPAC carrier driver, it's necessary to install also the IPAC carrier driver. Please refer to the IPAC carrier driver user manual for further information.

The standard file (I/O) functions (open, close, read, write and ioctl) provide the basic interface for opening and closing a file descriptor and for performing device I/O and control operations.

The TIP672 device driver includes the following functions:

- 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)
- TEWS TECHNOLOGIES IPAC carrier driver support.

To understand all features of this device driver, it is very important to read the controller manual, which is part of the engineering kit TIP672-EK.

2 Installation

The software is delivered on a PC formatted 3½" HD diskette.

The directory A:\TIP672-SW-72 contains the following files:

TIP672-SW-72.pdf	This manual in PDF format
TIP672-SW-72.tar	Device Driver and Example sources

The TAR archive TIP672-SW-72.tar contains the following files and directories:

tip672/tip672.c	Driver source code
tip672/tip672.h	Definitions and data structures for driver and application
tip672/tip672def.h	Definitions and data structures for the driver
tip672/tip672_info.c	Device information definition
tip672/tip672_info.h	Device information definition header
tip672/tip672.cfg	Driver configuration file include
tip672/tip672.import	Linker imports file for PowerPC platforms
tip672/Makefile	Device driver make file
tip672/Example/example.c	Example application source
tip672/Example/Makefile	Example make file

In order to perform a driver installation first extract the TAR file to a temporary directory then copy the following files to their target directories:

1. Create a new directory in the system drivers directory path /sys/drivers.xxx, where xxx represents the BSP that supports the target hardware.

   For example: /sys/drivers.pp_drm/tip672 or /sys/drivers.cpci_x86/tip672

2. Copy the following files to this directory:
   - tip672.c
   - tip672def.h
   - tip672.import
   - Makefile

3. Copy tip672.h to /usr/include/

4. Copy tip672_info.c to /sys/devices.xxx/ or /sys/devices if /sys/devices.xxx does not exist (xxx represents the BSP).

5. Copy tip672_info.h to /sys/dheaders/

6. Copy tip672.cfg to */sys/cfg.xxx/*, where xxx represents the BSP for the target platform

For example: /sys/cfg.ppc or /sys/cfg.x86 ....

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 *A:\CARRIER-SW-72* on the separate distribution diskette.

2.1 Device Driver Installation

The two methods of driver installation are as follows:

- Static Installation
- Dynamic Installation (only native LynxOS systems)

Both installation methods require the TEWS TECHNOLOGIES IPAC Carrier Driver. Please refer to the IPAC Carrier Driver User Manual for detailed information.

2.1.1 Static Installation

With this method, the driver object code is linked with the kernel routines and is installed during system start-up.

2.1.1.1 Build the driver object

1. Change to the directory /sys/drivers.xxx/tip672, where xxx represents the BSP that supports the target hardware.
2. To update the library /sys/lib/libdrivers.a enter:

   ```
   make install
   ```

2.1.1.2 Create Device Information Declaration

1. Change to the directory /sys/devices.xxx/ or /sys/devices if /sys/devices.xxx does not exist (xxx represents the BSP).
2. Add the following dependencies to the Makefile

   ```
   DEVICE_FILES_all = ... tip672_info.x
   ```

   And at the end of the Makefile

   ```
   tip672_info.o:$(DHEADERS)/tip672_info.h
   ```

3. To update the library /sys/lib/libdevices.a enter:

   ```
   make install
   ```

2.1.1.3 Modify the Device and Driver Configuration File

In order to insert the driver object code into the kernel image, an appropriate entry in file CONFIG.TBL must be created.

1. Change to the directory /sys/lynx.os/ respective /sys/bsp.xxx, where xxx represents the BSP that supports the target hardware.
2. Create an entry at the end of the file CONFIG.TBL

   Insert the following entry at the end of this file. Be sure that the necessary TEWS TECHNOLOGIES IPAC carrier driver is included **before** this entry.

   ```
   I:tip672.cfg
   ```

2.1.1.4 Rebuild the Kernel

1. Change to the directory /sys/lynx.os/ (/sys/bsp.xxx)
2. Enter the following command to rebuild the kernel:

   ```
   make install
   ```

3. Reboot the newly created operating system by the following command (not necessary for KDIs):

   ```
   reboot –aN
   ```

   The N flag instructs init to run mknod and create all the nodes mentioned in the new nodetab.

4. After reboot you should find the following new devices (depends on the device configuration): /dev/tip672_0, /dev/tip672_1, /dev/tip672_2, …

2.1.2 Dynamic Installation

This method allows you to install the driver after the operating system is booted. The driver object code is attached to the end of the kernel image and the operating system dynamically adds this driver to its internal structures. The driver can also be removed dynamically.

2.1.2.1 Build the driver object

1. Change to the directory /sys/drivers.xxx/tip672, where xxx represents the BSP that supports the target hardware.
2. To make the dynamic link-able driver enter :

   ```
   make
   ```

2.1.2.2 Create Device Information Declaration

1. Change to the directory /sys/devices.xxx/ or /sys/devices if /sys/devices.xxx does not exist (xxx represents the BSP).
2. To create a device definition file for the major device (this works only on native system)

   ```
   make t672info
   ```

3. To install the driver enter:

   ```
   drinstall –c tip672.obj
   ```

   If successful, drinstall returns a unique <driver-ID>
4. To install the major device enter:

   ```
   devinstall –c –d <driver-ID> t672info
   ```

   The <driver-ID> is returned by the drinstall command
5. To create nodes for the devices enter:

   ```
   mknod /dev/tip672_0 c <major_no> 0
   mknod /dev/tip672_1 c <major_no> 1
   mknod /dev/tip672_2 c <major_no> 2
   ...
   ```

   The <major_no> is returned by the devinstall command.

If all steps are successful completed the TIP672 is ready to use.

2.1.2.3 Uninstall dynamic loaded driver

To uninstall the TIP672 device enter the following commands:

```
devinstall –u –c <device-ID>
drinstall –u <driver-ID>
```

2.1.3 Configuration File: CONFIG.TBL

The device and driver configuration file CONFIG.TBL contains entries for device drivers and its major and minor device declarations. Each time the system is rebuild, the config utility read this file and produces a new set of driver and device configuration tables and a corresponding nodetab.

To install the TIP672 driver and devices into the LynxOS system, the configuration include file tip672.cfg must be included in the CONFIG.TBL (see also 2.1.1.3).

The file tip672.cfg on the distribution disk contains the driver entry (*C:tip672:*\....) and a major device entry (*D:TIP672:t672info::*) with 9 minor device entries (*“N: tip672_0:0”, ..., “N: tip672_8:8”*).

If the driver should support more than 9 minor devices because more than 9 TIP672 are plugged, additional minor device entries must be added. To create the device node */dev/tip672_9* the line *N:tip672_9:9* must be added at the end of the file tip672.cfg. For the next node a minor device entry with 10 must be added and so on.

This example shows a driver entry with one major device and nine minor devices:

```
#    Format:
#    C:driver-name:open:close:read:write:select:control:install:uninstall
#    D:device-name:info-block-name:raw-partner-name
#    N:node-name:minor-dev

C:tip672:\
     :t672open:t672close:t672read:t672write:\
     ::t672ioctl:t672install:t672uninstall
D:TIP672:t672info::
N:tip672_0:0
N:tip672_1:1
N:tip672_2:2
N:tip672_3:3
N:tip672_4:4
N:tip672_5:5
N:tip672_6:6
N:tip672_7:7
N:tip672_8:8
```

The configuration above creates the following node in the /dev directory.

/dev/tip672_0 ... /dev/tip672_8

3 TIP672 Device Driver Programming

LynxOS system calls are all available directly to any C program. They are implemented as ordinary function calls to "glue" routines in the system library, which trap to the OS code.

Note that many system calls use data structures, which should be obtained in a program from appropriate header files. Necessary header files are listed with the system call synopsis.

3.1 open()

NAME

open() - open a file

SYNOPSIS

#include <sys/file.h>
#include <sys/types.h>
#include <fcntl.h>

int open (char *path, int oflags[, mode_t mode])

DESCRIPTION

Opens a file (TIP672 device) named in path for reading and writing. The value of oflags indicates the intended use of the file. In case of a TIP672 devices oflags must be set to O_RDWR to open the file for both reading and writing.

The mode argument is required only when a file is created. Because a TIP672 device already exists this argument is ignored.

EXAMPLE

```
int  fd


fd = open ("/dev/tip672_0", O_RDWR);
```

RETURNS

Open returns a file descriptor number if successful or 1 on error. The global variable *errno* contains the detailed error code.

3.2 close()

NAME

close() – close a file

SYNOPSIS

int close(int fd)

DESCRIPTION

This function closes an opened device associated with the valid file descriptor handle fd.

EXAMPLE

```
int  result;

result = close(fd);
```

RETURNS

Close returns 0 (OK) if successful, or –1 on error. The global variable errno contains the detailed error code.

SEE ALSO

LynxOS System Call - close()

3.3 read()

NAME

read() - read from a file

SYNOPSIS

#include <tip672.h>

int read (int fd, char *buff, int count)

DESCRIPTION

This function attempts to read the input registers of the TIP672 associated with the file descriptor *fd* into a 32-bit unsigned long addressed by *buffer*. The argument *count* specifies the length of the buffer and must be set to the length of unsigned long

The unsigned long variable stores the content of the line input register in the lower 24 bit, where bit 2^0 corresponds to INPUT 1 and bit 2^{23} corresponds to INPUT 24. The upper 8 bit of the unsigned long variable are always read as 0.

EXAMPLE

```
int  fd;
int  result;
unsigned long value;


result = read(fd, (char*)& value, sizeof(unsigned long));

if(result != sizeof(unsigned long)) {
     /* handle read error */
}
```

RETURNS

When read succeeds, the size of the read buffer is returned. If read fails, -1 (SYSERR) is returned.

On error, errno will contain a standard read error code (see also LynxOS System Call – read) or one of the following TIP672 specific error codes:

ENXIO	Illegal device
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

LynxOS System Call - read()

3.4 write()

NAME

write() – write to a file

SYNOPSIS

int write (int fd, char *buff, int count)

DESCRIPTION

This function attempts to write to the output registers of the TIP672 associated with the file descriptor *fd* from a unsigned long value pointed by buffer. The argument count specifies the length of the buffer and must be set to the length of an unsigned long.

The lower 24 bit of the unsigned long variable corresponds to the 24 output lines.

EXAMPLE

```
int  fd;
int  result;
unsigned long value;


// set OUTPUT 1,15 and 24 to logic high
value = 0x804001;
result = write(fd, &value, sizeof(unsigned long));

if(result != sizeof(unsigned long)) {
     /* handle write error */
}
```

RETURNS

When write succeeds, the size of the write buffer is returned. If write fails, -1 (SYSERR) is returned.

On error, errno will contain a standard write error code (see also LynxOS System Call – write) or the following TIP672 specific error code:

ENXIO	Illegal device
EINVAL	Invalid argument. This error code is returned if the size of the message buffer is too small.

SEE ALSO

LynxOS System Call - write()

3.5 ioctl()

NAME

ioctl() - I/O device control

SYNOPSIS

#include <ioctl.h>
#include <tip672.h>

int ioctl (int fd, int request, char *arg)

DESCRIPTION

ioctl provides a way of sending special commands to a device driver. The call sends the value of request and the pointer arg to the device associated with the descriptor fd.

The following ioctl codes are defined in TIP672.h :

Value	Meaning
T672_SET_DIR	Set direction of I/O lines
T672_ENA_EXCLK	Enable simultaneous update feature
T672_DIS_EXCLK	Disable simultaneous update feature
T672_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.

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.

The TIP672 ioctl function returns always standard error codes.

SEE ALSO

LynxOS System Call – ioctl() for detailed description of possible error codes.

3.5.1 T672_SET_DIR

NAME

T672_SET_DIR - Set direction of I/O lines

DESCRIPTION

With this ioctl function each of the 24 I/O line may be individually set as input or output. Bit 0 of the direction mask corresponds to line 1, bit 1 to line 2 and so on. To set a line to be input, set the corresponding bit in the mask to 1.

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

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

EXAMPLE

```
int  fd;
int  result;
unsigned long direction;


//   Set line 1..12 as input and line 13..24 as output
direction = 0x000FFF;

result = ioctl(fd, T672_SET_DIR, (char*)&direction);

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

3.5.2 T672_ENA_EXCLK

NAME

T672_ENA_EXCLK - Enable simultaneous update feature

DESCRIPTION

This ioctl function enables the simultaneous update feature of the TIP672. 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

```
int  fd;
int  result;
unsigned long value;


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

result = ioctl(fd, T672_ENA_EXCLK, (char*)&value);

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

RETURNS

EINVAL	Invalid parameter

3.5.3 T672_DIS_EXCLK

NAME

T672_DIS_EXCLK - Disable simultaneous update feature

DESCRIPTION

This ioctl function disables the simultaneous update feature of the TIP672. The argument *argp* is not needed and can be omitted.

After driver startup the simultaneous update feature is disabled.

EXAMPLE

```
int  fd;
int  result;


result = ioctl(fd, T672_DIS_EXCLK, (char*)NULL);

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

3.5.4 T672_WAIT_TRANS

NAME

T672_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 contents of the input register to the caller.

A pointer to the callers parameter buffer (*T672_TRANS_PAR*) is passed by the parameter *argp* to the driver.

The *T672_TRANS_PAR* structure has the following layout:

```
typedef struct {
        unsigned long     selectMask;
        unsigned long     polarityMask;
        long              timeout;
        unsigned long     statusMask;
        unsigned long     inputReg;
} T672_TRANS_PAR, *PT672_TRANS_PAR;
```

unsigned long selectMask

This parameter contains a bit mask to select a certain bit position or a group of bits for an input transition detection. Bits 2^0 to 2^{23} correspond to INPUT 1 to INPUT 24. A certain input line can be selected by setting the corresponding bit position to 1. A certain input line can occupied only ones. If you try to select an input line which is already occupied by other requests you will got the error code *EBADSLT*.

unsigned long polarityMask

This parameter defines the active transition for the selected input lines. Bits 2^0 ... 2^{23} correspond to INPUT 1 ... INPUT 24. To generate an event for a positive transition (0->1) set the corresponding bit to 1. For a negative transition (1->0) set the corresponding bit to 0. Only selected bits (*selectMask*) are relevant for the polarity setting.

long timeout

Specifies the amount of time (in ticks) the caller is willing to wait for the occurrence of the requested transition. A value of -1 means wait indefinitely or no timeout.

unsigned long statusMask

This parameter receives the contents of the TIP672 interrupt status register and 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.

unsigned long inputReg

This parameter receives the contents of the line input register after the requested event has occurred. Please note that the input register is not latched with the interrupt and depending on the interrupt latency the read to the input register is delayed.

EXAMPLE

```
int  fd;
int  result;
T672_TRANS_PAR par;


// Wait for a (0->1) transition at INPUT 24 OR a (1->0) transition
// at INPUT 1.
// The request times out after 1000 ticks.
par.select_mask    =    0x800001;
par.polarity_mask  =    0x800000;
par.timeout        =    1000;

result = ioctl(fd, T672_WAIT_TRANS, (char*)&par);

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

RETURNS

EINVAL	At least one of the chosen input lines is already waiting for an event..
ETIMEDOUT	The allowed time to finish the request is elapsed.
EINTR	Interrupted system call (probably by a signal).
EBUSY	There is no free entry in the event queue. You should increase the number of possible event entries. (MAX_REQUESTS in tip673def.h)

4 Debugging and Diagnostic

If your installed IPAC port driver (e.g. tip672) doesn't find any devices although the IPAC is properly plugged on a carrier port, it's interesting to know what's going on in the system.

Usually all TEWS TECHNOLOGIES device driver announced significant event or errors via the device driver routine kkprintf(). To enable the debug output you must define the macro DEBUG in the device driver source files (e.g. carrier_class.c, carrier_tews_pci.c, tip672.c,...).

The debug output should appear on the console. If not please check the symbol KKPF_PORT in uparam.h. This symbol should be configured to a valid COM port (e.g. SKDB_COM1).

The following output appears at the LynxOS debug console if the carrier and IPAC driver starts:

```
TEWS TECHNOLOGIES - IPAC Carrier Class Driver version 1.0.0 (2003-11-28)
TEWS TECHNOLOGIES - VME Carrier version 1.0.0 (2003-12-07)
IPAC_CC  :  IPAC (Manuf-ID=B3, Model#=34) recognized @ slot=1 carrier=<TEWS TEC>
TIP672 - 24 differential I/O Lines with Interrupts version 1.0.0 (2003-12-13)
TIP672  :  Probe new TIP672 mounted on <TEWS TECHNOLOGIES - VME Carrier> at slot B
```

If you can't solve the problem by yourself, please contact TEWS TECHNOLOGIES with a detailed description of the error condition, your system configuration and the debug outputs.