The Embedded I/O Company

TEWS TECHNOLOGIES

TPCI868-SW-82

Linux Device Driver

16 Channel Serial Interface RS232

Version 1.0.x

User Manual

Issue 1.0.0

April 2006

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

TPCI868-SW-82

16 Channel Serial Interface RS232

Linux Device Driver

Issue	Description	Date
1.0.0	First Issue	April 07, 2006

Table of Contents

1 Introduction

The TPCI868-SW-82 Linux device driver is a full-duplex serial driver which allows the operation of a TPCI868 module on Linux (Kernel 2.4.x+ and 2.6.x+) operating systems.

The TPCI868-SW-82 device driver based on the standard Linux serial device driver and supports all standard terminal functions (TERMIOS).

Supported features:

- Extended baudrates up to 921600 BAUD.
- Each channel has a 64 Byte transmit and receive hardware FIFO
- Programmable trigger level for transmit and receive FIFO.
- Hardware (RTS/CTS) and software flow control (XON/XOFF) direct controlled by the serial controller. The advantage of this feature is that the transmission of characters will immediately stop as soon as a complete character is transmitted and not when the transmit FIFO is empty for handshake under software control. This will greatly improve flow control reliability.
- Designed as Linux kernel module with dynamically loading.
- Supports shared IRQ's.
- Build on new style PCI driver layout
- Creates a TTY device (Kernel 2.4.x) with dynamically allocated or fixed major device numbers.
- Support for automatic device node creation
- IOCTL function for a Built-In-Self-Test

The TPCI868-SW-82 device driver supports the modules listed below:

TPCI868-10	16 Channel Serial Interface RS232	(PCI)

To get more information about the features and use of TPCI868 device it is recommended to read the manuals listed below.

TPCI868 User manual

TPCI868 Engineering Manual

ST16C654 UART Hardware Manual

In case of difficulties during installation please contact TEWS TECHNOLOGIES.

2 Installation

The directory TPCI868-SW-82 on the distribution media contains the following files:

TPCI868-SW-82-1.0.0.pdf	This manual in PDF format
TPCI868-SW-82-SRC.tar.gz	GZIP compressed archive with driver source code
Release.txt	Release information

The GZIP compressed archive TPCI868-SW-82-SRC.tar.gz contains the following files and directories:

example/Makefile	Example application makefile
example/tpci868example.c	Send and receive example application
example/tpci868setspeed.c	Speed configuration example application
example/tpci868bist.c	Example for using Built-In-Self-Test
HAL/	Hardware abstraction layer driver needed for all kernel versions
HAL/Makefile	HAL driver makefile
HAL/tpci868hal.c	HAL driver source file
HAL/tpci868haldef.h	HAL driver private header file
HAL/include/tpxxxhwdep.c	HAL low level WINNT style hardware access functions source file
HAL/include/tpxxxhwdep.h	HAL access functions header file
Serial/	UART driver directory
Serial/2.4.x	Kernel 2.4.x sources directory
Serial/2.4.x/Makefile	Serial driver makefile
Serial/2.4.x/tpci868serial.c	Serial driver source file
Serial/2.4.x/tpci868serialdef.h	Serial driver private header file
Serial/2.6.x	Kernel 2.6.x sources directory
Serial/2.6.x/Makefile	Serial driver makefile
Serial/2.6.x/tpci868serial.c	Serial driver source file
Serial/2.6.x/tpci868serialdef.h	Serial driver private header file
Serial/makenode	Shell script to create devices nodes manually
tpci868def.h	Driver private header file
tpci868.h	User application header file

In order to perform an installation, extract all files of the archive TPCI868-SW-82-SRC.tar.gz to the desired target directory.

- Login as *root* and change to the target directory
- Copy tpci868.h to */lib/modules/<version>/build/include* and */usr/include*

2.1 Build and install the device driver

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

  # make install

- Change to the Serial/<version> target directory
- To create and install the SERIAL driver in the module directory */lib/modules/<version>/misc* enter:
  # make install
- To update module dependencies enter:

  # 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

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 tpci868serialdrv

- 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,* which resides in Serial/ directory, to do this. If your kernel has enabled a device file system (DEVFS or UDEV) then 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 compatible channel found. The first channel of the first PCI module can be accessed with device node /dev/ttySTPCI868_0, the second channel with device node /dev/ttySTPCI868_1 and so on. The assignment of device nodes to physical PCI modules depends on the search order of the PCI bus driver.

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 –r tpci868serialdrv

If your kernel has enabled a device file system like DEVFS or UDEV, all /dev/ttySTPCI868_* 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 "*tpci868serialdrv: 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

This paragraph is only for Linux kernels without an installed device file system like DEVFS or UDEV.

The released TPCI868 driver use dynamic allocation of major device numbers. If this isn't suitable for the application it's possible to define a major number for the *TTY* driver.

To change the major number edit the file Serial/<version>/tpci868serial.c, change the following symbol to appropriate values and enter *make install* to create a new driver.

TPCI868_TTY_MAJOR	Defines the value for the terminal device. Valid numbers are in range between 0 and 255. A value of 0 means dynamic number allocation.

Example:

```
#define TPCI868_TTY_MAJOR       122
```

Be sure that the desired major number isn't used by other drivers. Please check */proc/devices* to see which numbers are free.

Keep in mind that's necessary to create new device nodes if the major number for the TPCI868 driver has changed and the makenode script isn't used.

2.6 FIFO configuration

After installation of the TPCI868 Device Driver the trigger level for transmit and receive FIFO are set to their default values.

Default values are:

Receive FIFO	Transmit FIFO
56	16

The configuration of the FIFO trigger level is used for all TPCI868 devices in common.

To change the trigger levels edit the file *HAL/tpci868haldef.h,* change the following symbols to appropriate values and enter ***make install*** to create a new driver.

TPCI868_RX_TRG_DEF Define the trigger level for the receiver FIFO of a TPCI868 with ST16C654 controller. Valid trigger levels are:
UART_FCR_R_TRIGGER_8
UART_FCR_R_TRIGGER_16
UART_FCR_R_TRIGGER_56
UART_FCR_R_TRIGGER_60

TPCI868_TX_TRG_DEF Define the trigger level for the transmitter FIFO of a TPCI868 with ST16C654 controller. Valid trigger levels are:
UART_FCR_T_TRIGGER_8
UART_FCR_T_TRIGGER_16
UART_FCR_T_TRIGGER_32
UART_FCR_T_TRIGGER_56

Please refer to the User Manual of the ST16C654 controller to get more information how to customize suitable FIFO trigger level.

3 Device Driver Programming

The TPCI868-SW-82 driver loosely bases on the standard Linux terminal driver. Due to this way of implementation the driver interface and functionality is compatible to the standard Linux terminal driver.

Please refer to the TERMIOS man page and driver programming related man pages for more information about serial driver programming.

3.1 Simple Programming example

This example program opens the first serial channel of a TPCI868 compatible PMC for read/write. After the device is open it writes a “Hello World” string to the device and receives up 80 bytes from the serial channel.

```
main()
{
  int fd;
  int count;
  char buffer[81];

  /* open the desired PMC device channel*/
  fd = open( “/dev/ttySTPCI868_0”, O_RDWR | O_NOCTTY);

  if (fd < 0) exit(-1);

  /* write data to the certain channel */
  count = write(fd, “Hello World\n”, 12);
  printf(“%d bytes written\n”, count);

  /* read up to 80 bytes from the device */
  count = read(fd, buffer, 80)
  if (count < 0) {
    printf(“read error\n”);
  }
  else {
    buffer[count] = 0;
    printf(“%d bytes read <%s>\n”, count, buffer);
  }

  close(fd);
}
```

The source files *tpci868example.c* and *tpci868setspeed.c* contains additional programming examples.

3.2 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 *tpci868.h*:

Value	Meaning
TPCI868_IOCQ_BIST	Start Built-In-Self-Test

See below for more detailed information on each control code.

To use these TPCI868 specific control codes the header file *tpci868.h* must be included in the application.

RETURNS

On success, zero is returned. In 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 TPCI868 driver always returns standard Linux error codes.

SEE ALSO

ioctl man pages

3.2.1 TPCI868_IOCQ_BIST

NAME

TPCI868_IOCQ_BIST – Start Built-In-Self-Test

DESCRIPTION

The TPCI868 driver supports a special IOCTL function for testing module hardware and for system diagnostic. The optional argument can be omitted for this ioctl function.

The functionality is called Built-In-Self-Test or BIST. With BIST you can test each channel of all your modules separately. There are three different test classes. First is a line test, second an interrupt test and the last a data integrity test. All tests run with local channel loopback enabled, so you don't need an external cable connection.

For a detailed description of the loopback wiring please refer to the ST16C654 manual and see the description of *Internal Loopback*.

The line test contains a test of all modem lines, as you can see RTS and CTS, DTR and DSR, OP1 and RI and finally OP2 and CD. Only the static states for both electrical levels are tested on each sender – receiver line pair.

For testing interrupts the BIST transmits a test buffer with known data and size. All data should be received on same channel during internal loopback. If not, there is an interrupt error. The buffer size is 1024 BYTE. The baudrate has to be set through the standard terminal IOCTL functions.

The last test verifies received data to assert data integrity.

This function tests all internal I/O lines of the controller, also if they are not used for interfacing. DTR, DSR, RI and CD are not used on the TPCI868.

EXAMPLE

```
/* Start Built-In Selftest, */
result = ioctl(tty1, TPCI868_IOCQ_BIST, NULL);

if (result) printf("Error during Built-In Selftest <%d, 0x%08X>!\n",
  result, result);
if (result < 0)
{
  printf("ERRNO %d - %s\n", errno, strerror(errno));
}
else if (result > 0)
{
  if (result & TPCI868ERTSCTS)
    printf("RTS/CTS line broken!\n");
  if (result & TPCI868_EDTRDSR)
    printf("DTR/DSR line broken!\n");
  if (result & TPCI868_ERI)
    printf("OP1/RI line broken!\n");
  if (result & TPCI868_ECD)
    printf("OP2/DCD line broken!\n");
  if (result & TPCI868_EDATA)
    printf("Data integrity test failed!\n");
}else
  printf("INFO: Port %s successfully tested.\n", DevName);
```

RETURNS

If return value is >0 one of three tests failed. Use the following flags to get a detailed error description.

TPCI868_ERTSCTS	If set RTS/CTS line broken.
TPCI868_EDTRDSR	If set DTR/DSR line broken.
TPCI868_ERI	If set OP1/RI line broken.
TPCI868_ECD	If set OP2/CD line broken.
TPCI868_EDATA	Data integrity test failed. No correct transmission possible.

ERRORS

ETIME	A timeout occurred during wait, interrupts do not work correctly.
EAGAIN	Your task should never been blocked. Change it to use the Built-In-Self-Test.
ERESTARTSYS	Interrupted by external signal.

4 Diagnostic

If the TPCI868 driver does not work properly it is helpful to get some status information from the driver respective kernel.

The Linux */proc* file system provides information about kernel, resources, driver, devices and so on. The following screen dumps displays information of a correct running TPCI868 driver (see also the proc man pages).

The output shown below may differ on different system depending on the distribution, kernel version and target system. (The example used 2.6.x kernel and a 2 CPU board)

```
# cat /proc/tty/driver/tpci868serial
serinfo:1.0 driver revision:
0: uart:ST16C654 mmio:0xFF5FE800 irq:193 tx:0 rx:0
1: uart:ST16C654 mmio:0xFF5FE808 irq:193 tx:0 rx:0
2: uart:ST16C654 mmio:0xFF5FE810 irq:193 tx:0 rx:0
3: uart:ST16C654 mmio:0xFF5FE818 irq:193 tx:0 rx:0
4: uart:ST16C654 mmio:0xFF5FE820 irq:193 tx:0 rx:0
5: uart:ST16C654 mmio:0xFF5FE828 irq:193 tx:0 rx:0
6: uart:ST16C654 mmio:0xFF5FE830 irq:193 tx:0 rx:0
7: uart:ST16C654 mmio:0xFF5FE838 irq:193 tx:0 rx:0
8: uart:ST16C654 mmio:0xFF5FE840 irq:193 tx:0 rx:0
9: uart:ST16C654 mmio:0xFF5FE848 irq:193 tx:0 rx:0
10: uart:ST16C654 mmio:0xFF5FE850 irq:193 tx:0 rx:0
11: uart:ST16C654 mmio:0xFF5FE858 irq:193 tx:0 rx:0
12: uart:ST16C654 mmio:0xFF5FE860 irq:193 tx:0 rx:0
13: uart:ST16C654 mmio:0xFF5FE868 irq:193 tx:0 rx:0
14: uart:ST16C654 mmio:0xFF5FE870 irq:193 tx:0 rx:0
15: uart:ST16C654 mmio:0xFF5FE878 irq:193 tx:0 rx:0

# cat /proc/tty/drivers
/dev/tty             /dev/tty        5       0 system:/dev/tty
/dev/console         /dev/console    5       1 system:console
/dev/ptmx            /dev/ptmx       5       2 system
/dev/vc/0            /dev/vc/0       4       0 system:vtmaster
tpci868serial        /dev/ttySTPCI868_ 254   0-127 serial
serial               /dev/ttyS       4       64-71 serial
pty_slave            /dev/pts      136       0-1048575 pty:slave
pty_master           /dev/ptm      128       0-1048575 pty:master
unknown              /dev/tty        4       1-63 console
```

(With at least one opened TPCI868 devices)

```
# cat /proc/interrupts
           CPU0       CPU1
  0:     580333     627739    IO-APIC-edge  timer
  1:          0         11    IO-APIC-edge  i8042
  2:          0          0          XT-PIC  cascade
  8:          0          1    IO-APIC-edge  rtc
  9:          0          0   IO-APIC-level  acpi
 12:          0         58    IO-APIC-edge  i8042
 14:       2419       8436    IO-APIC-edge  ide0
169:      85901      60646   IO-APIC-level  radeon@PCI:1:0:0
177:          0          0   IO-APIC-level  uhci_hcd
185:       4243         28   IO-APIC-level  uhci_hcd, eth0
193:      72585      32422   IO-APIC-level  libata, ehci_hcd, ohci_hcd,
ohci_hcd, TPCI868
NMI:          0          0
LOC:    1207923    1207922
ERR:          0
MIS:          0

# lspci –v
...
02:07.0 Multiport serial controller: TEWS Datentechnik GmBH: Unknown device
3364 (rev 0a)
        Subsystem: TEWS Datentechnik GmBH: Unknown device 000a
        Flags: medium devsel, IRQ 193
        Memory at ff5fec00 (32-bit, non-prefetchable)
        I/O ports at a880 [size=128]
        Memory at ff5fe800 (32-bit, non-prefetchable) [size=256]
        Memory at ff5fe400 (32-bit, non-prefetchable) [size=16]
...
```