The Embedded I/O Company

TEWS TECHNOLOGIES

TPMC860-SW-82

Linux Device Driver

4 Channel Isolated Serial Interface RS232

Version 1.4.x

User Manual

Issue 1.4.3

April 2010

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

TPMC860-SW-82

Linux Device Driver

4 Channel Isolated Serial Interface RS232

Issue	Description	Date
1.0	First Issue	September 2001
1.1	General Revision	March 2004
1.2.0	Kernel 2.6.x Support	March 2005
1.3.0	Built-In-Self-Test added	May 23, 2005
1.4.0	Introduction and installation section (file list) modified	March 29, 2006
1.4.1	File list modified, New Address TEWS LLC, general revision	November 6, 2006
1.4.2	File list corrected	January 20, 2009
1.4.3	Address TEWS LLC removed	April 14, 2010

Table of Contents

1 Introduction

The TPMC860-SW-82 Linux device driver is a full-duplex serial driver which allows the operation of a TPMC860 module on Linux (Kernel 2.4.x+ and 2.6.x+) operating systems.

The TPMC860-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 460800 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 and dialout device (Kernel 2.4.x) with dynamically allocated or fixed major device numbers.
- DEVFS support for automatic device node creation
- IOCTL function for a Built-In-Self-Test

The TPMC860-SW-82 device driver supports the modules listed below:

TPMC860-10	4 Channel Isolated Serial Interface RS232	(PMC)

To get more information about the features and use of TPMC860 device it is recommended to read the manuals listed below.

TPMC860 User manual

TPMC860 Engineering Manual

ST16C554 UART Hardware Manual

In case of difficulties during installation please contact TEWS TECHNOLOGIES.

2 Installation

The directory TPMC860-SW-82 on the distribution media contains the following files:

TPMC860-SW-82-1.4.3.pdf	This manual in PDF format
TPMC860-SW-82-SRC.tar.gz	GZIP compressed archive with driver source code
Release.txt	Release information
ChangeLog.txt	Release history

The GZIP compressed archive TPMC860-SW-82-SRC.tar.gz contains the following files and directories:

example/Makefile	Example application makefile
example/tpmc860example.c	Send and receive example application
example/tpmc860setspeed.c	Speed configuration example application
example/tpmc860bist.c	Example for using Built-In-Self-Test
hal/	Hardware abstraction layer driver needed for all kernel versions
hal/Makefile	HAL driver makefile
hal/tpmc860hal.c	HAL driver source file
hal/tpmc860haldef.h	HAL driver private header file
include/tpmodule.c	Driver independent library
include/tpmodule.h	Driver independent library header file
include/config.h	Driver independent library header file
include/tpxxxhwdep.c	HAL low level WINNT style hardware access functions source file
include/tpxxxhwdep.h	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/tpmc860serial.c	Serial driver source file
Serial/2.4.x/tpmc860serialdef.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/tpmc860serial.c	Serial driver source file
Serial/2.6.x/tpmc860serialdef.h	Serial driver private header file
Serial/makenode	Shell script to create devices nodes manually
tpmc860def.h	Driver private header file
tpmc860.h	User application header file

In order to perform an installation, extract all files of the archive TPMC860-SW-82-SRC.tar.gz to the desired target directory.

- Login as *root* and change to the target directory
- Copy tpmc860.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

  For Linux kernel 2.6.x, there may be compiler warnings claiming some undefined tpmc860_hal_* symbols. These warnings are caused by the HAL driver, which is unknown during compilation of this SERIAL driver. The warnings can be ignored.

- To update module dependencies enter:

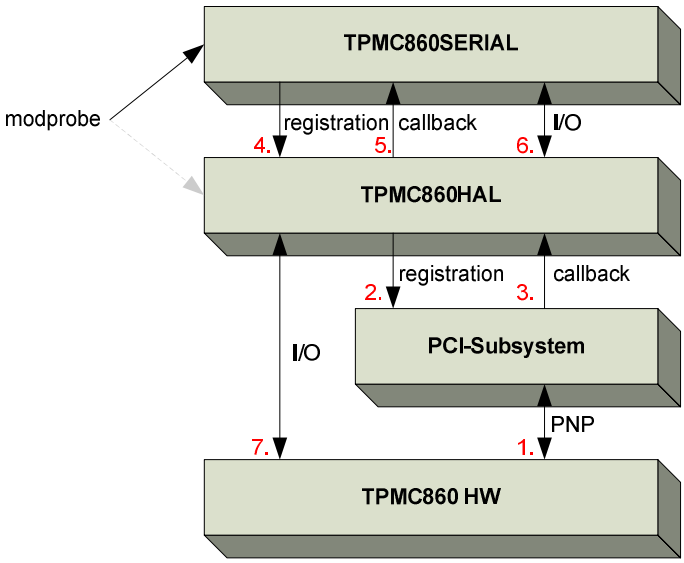
  # depmod -aq

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 tpmc860serialdrv

- 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 the device file system (devfs) 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 PMC module can be accessed with device node /dev/ttySTPMC860_0, the second channel with device node /dev/ttySTPMC860_1 and so on. The assignment of device nodes to physical PMC 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 tpmc860serialdrv

If your kernel has enabled devfs, all /dev/ttySTPMC860_* 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 "*tpmc860serialdrv: 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 DEVFS installed.

The released TPMC860 driver use dynamic allocation of major device numbers. If this isn't suitable for the application it's possible to define a major number separately for the *TTY* and *CUA* driver.

To change the major number edit the file Serial/<version>/tpmc860serial.c, change the following symbols to appropriate values and enter *make install* to create a new driver.

TPMC860_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.
TPMC860_CUA_MAJOR	Defines the value for the dialout device. Valid numbers are in range between 0 and 255. A value of 0 means dynamic number allocation.

Example:

```
#define TPMC860_TTY_MAJOR       122
#define TPMC860_CUA_MAJOR       123
```

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 it is necessary to create new device nodes if the major number for the TPMC860 driver has changed and the makenode script isn't used.

2.6 FIFO configuration

After installation of the TPMC860 Device Driver the trigger level for the 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 TPMC860 devices in common.

To change the trigger levels edit the file *HAL/tpmc860haldef.h,* change the following symbols to appropriate values and enter ***make install*** to create a new driver.

TPMC860_RX_TRG_DEF Define the trigger level for the receiver FIFO of a TPMC860 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

TPMC860_TX_TRG_DEF Define the trigger level for the transmitter FIFO of a TPMC860 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 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/ttySTPMC860_0”, O_RDWR);
if (fd < 0)
{
  /* handle open error conditions */
}
```

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. For more information about open error codes, see the *GNU C Library description – Low-Level Input/Output*.

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 ioctl()

NAME

ioctl() device control functions

SYNOPSIS

#include <sys/ioctl.h>
#include <tpmc860.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 *tpmc860.h*:

Value	Meaning
TPMC860_IOCQ_BIST	Start Built-In-Self-Test

See below for more detailed information on each control code.

To use these TPMC860 specific control codes the header file *tpmc860.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 TPMC860 driver always returns standard Linux error codes.

SEE ALSO

ioctl man pages

3.3.1 TPMC860_IOCQ_BIST

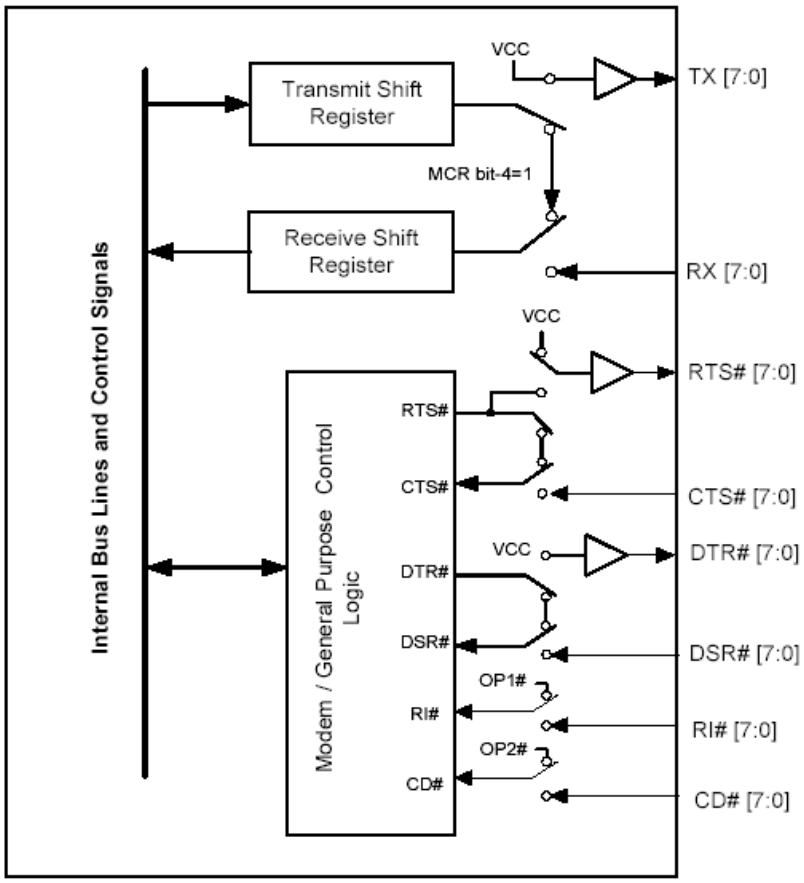
NAME

TPMC860_IOCQ_BIST – Start Built-In-Self-Test

DESCRIPTION

The TPMC860 driver (version 1.1.0 and higher) 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.



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.

EXAMPLE

```
#include <tpmc860.h>
int tty1;
int result;

/* Start Built-In Selftest, */
result = ioctl(tty1, TPMC860_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 & TPMC860_ERTSCTS)
    printf("RTS/CTS line broken!\n");
  if (result & TPMC860_EDTRDSR)
    printf("DTR/DSR line broken!\n");
  if (result & TPMC860_ERI)
    printf("OP1/RI line broken!\n");
  if (result & TPMC860_ECD)
    printf("OP2/DCD line broken!\n");
  if (result & TPMC860_EDATA)
    printf("Data integrity test failed!\n");
}else
  printf("INFO: Port successfully tested.\n");
```

RETURNS

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

TPMC860_ERTSCTS	If set RTS/CTS line broken.
TPMC860_EDTRDSR	If set DTR/DSR line broken.
TPMC860_ERI	If set OP1/RI line broken.
TPMC860_ECD	If set OP2/CD line broken.
TPMC860_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 Device Driver Programming

The TPMC860-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.

4.1 Simple Programming example

This example program opens the first serial channel of a TPMC860 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.

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

   /* open the desired PMC device channel*/
   fd = open( “/dev/ttySTPMC860_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 *tpmc860example.c* and *tpmc860setspeed.c* contains additional programming examples.

5 Diagnostic

If the TPMC860 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 TPMC860 driver (see also the proc man pages).

```
# cat /proc/tty/driver/tpmc860serial
TEWS TECHNOLOGIES - TPMC860 UART driver (Kernel 2.4.x): 1.4.1 revision:
2006-11-06
0: uart:ST16C654 port:D08EFF40 irq:10 tx:0 rx:0
1: uart:ST16C654 port:D08EFF48 irq:10 tx:0 rx:0
2: uart:ST16C654 port:D08EFF50 irq:10 tx:0 rx:0
3: uart:ST16C654 port:D08EFF58 irq:10 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
tpmc860serial        /dev/cuaTPMC860_%d  253  0-127 serial:callout
tpmc860serial        /dev/ttySTPMC860_%d 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

# cat /proc/interrupts
           CPU0
  0:     682903          XT-PIC  timer
  1:          6          XT-PIC  keyboard
  2:          0          XT-PIC  cascade
  8:          1          XT-PIC  rtc
 10:      96784          XT-PIC  ehci-hcd, TPMC860, TPMC860, TPMC860,
TPMC860
 11:      16339          XT-PIC  usb-uhci, usb-uhci, eth0
 12:         49          XT-PIC  PS/2 Mouse
 14:      33994          XT-PIC  ide0
 15:          8          XT-PIC  ide1
NMI:          0
ERR:          1
```

```
# lspci –v
...
00:11.0 Multiport serial controller: TEWS Datentechnik GmBH: Unknown device
  035c (rev 0a)
        Subsystem: TEWS Datentechnik GmBH: Unknown device 000a
        Flags: medium devsel, IRQ 11
        Memory at cfffde80 (32-bit, non-prefetchable)
        I/O ports at d000 [size=128]
        Memory at cfffde40 (32-bit, non-prefetchable) [size=64]
...
```