The Embedded I/O Company

TEWS TECHNOLOGIES

TPMC861-SW-82

Linux Device Driver

4 Channel Isolated Serial Interface

RS422/RS485

Version 1.4.x

User Manual

Issue 1.4.0

August 2013

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

TPMC861-SW-82

Linux Device Driver

4 Channel Isolated Serial Interface

RS422/RS485

Issue	Description	Date
1.0	First Issue	September 2001
1.1	General Revision	March 2004
1.2.0	Kernel 2.6.x Support	May 2, 2005
1.2.1	depmod for driver installation added	October 13, 2005
1.3.0	ELinOS makefiles removed from file list, UDEV support added, ChangeLog.txt added to file list	August 16, 2006
1.3.1	File list modified, general revision	May 27, 2008
1.3.2	New top level Makefile and diagnostic info for kernel 2.6	August 6, 2010
1.3.3	File list changed	December 22, 2011
1.4.0	New functions configuring user baudrate and reading actual configured baudrate	August 12, 2013

Table of Contents

1 Introduction

The TPMC861 Linux device driver is a full-duplex serial driver which allows the operation of a TPMC861 serial PMC on Linux operating systems.

The TPMC861 device driver is based on the standard Linux serial device driver and supports all standard terminal functions (TERMIOS).

The TPMC861 device driver includes the following features:

- Extended baud rates up to 460800 Baud
- Each channel has a 128 byte transmit and receive 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.
- Direct support of different physical interfaces (RS-422, RS-485 half-duplex and full-duplex)
- Designed as Linux kernel module with dynamically loading
- Supports shared IRQ's
- Build on new style PCI driver layout
- Creates a TTY device ttyTPMC861 with dynamically allocated or fixed major device numbers.
- DEVFS and UDEV support for automatic device node creation
- IOCTL function for a Built-In-Self-Test

The TPMC861-SW-82 device driver supports the modules listed below:

TPMC861-10	4 Channel Isolated Serial Interface RS422/RS485	(PMC)

To get more information about the features and use of TPMC861 device it is recommended to read the manuals listed below.

TPMC861 User manual
TPMC861 Engineering Manual
XR16C864 UART Hardware Manual

2 Installation

The directory TPMC861-SW-82 on the distribution media contains the following files:

TPMC861-SW-82-1.4.0.pdf	This manual in PDF format
TPMC861-SW-82-SRC.tar.gz	GZIP compressed archive with driver source code
Release.txt	Release information
ChangeLog.txt	Release history

The GZIP compressed archive TPMC861-SW-82-SRC.tar.gz contains the following files and directories:

example/Makefile	Example application makefile
example/tpmc861example.c	Send and receive example application
example/tpmc861setspeed.c	Speed configuration example application
example/tpmc861bist.c	Example for using Built-In-Self-Test
hal/	Hardware abstraction layer driver needed for all kernel versions
hal/Makefile	HAL driver makefile
hal/tpmc861hal.c	HAL driver source file
hal/tpmc861haldef.h	HAL driver private header file
include/config.h	Driver independent library header file
include/tpmodule.h	Driver and kernel independent library header file
include/tpmodule.c	Driver and kernel independent library source file
include/tpxxxhwdep.h	Hardware abstraction library header file
include/tpxxxhwdep.c	Hardware abstraction library source file
serial/	UART driver directory
serial/Makefile	Serial driver makefile
serial/tpmc861serial.c	Serial driver source file
serial/tpmc861serialdef.h	Serial driver private header file
serial/2.4.x	Kernel 2.4.x sources directory
serial/2.4.x/Makefile	Serial driver makefile
serial/2.4.x/tpmc861serial.c	Serial driver source file
serial/2.4.x/tpmc861serialdef.h	Serial driver private header file
serial/makenode	Shell script to create devices nodes manually
tpmc861def.h	Driver private header file
tpmc861.h	User application header file
Makefile	Top-level Makefile

In order to perform an installation, extract all files of the archive TPMC861-SW-82-SRC.tar.gz to the desired target directory. The command 'tar -xzvf TPMC861-SW-82-SRC.tar.gz' will extract the files into the local directory.

- Login as *root* and change to the target directory
- Copy tpmc861.h */usr/include*

2.1 Build and install the Device Driver

- Login as *root*
- Change to the tpmc861 target directory
- To create and install the HAL and SERIAL driver in the module directory */lib/modules/<version>/misc* enter:

  # make install

- To update the device driver's module dependencies, enter:

  # **depmod -aq**

2.2 Uninstall the Device Driver

- Login as *root*
- Change to the tpmc861 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 tpmc861serialdrv

- 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, 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 PMC module can be accessed with device node /dev/ttySTPMC861_0, the second channel with device node /dev/ttySTPMC861_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 tpmc861serialdrv

If your kernel has enabled a device file system (devfs, udev, ...), all /dev/ttySTPMC861_* nodes will be automatically removed from your file system after this.

Make sure that the driver isn't opened by any application program. If opened you will get the response "*tpmc861serialdrv: 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 a device file system installed.

The released TPMC861 driver uses 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>/tpmc861serial.c, change the following symbol to appropriate value and enter *make install* to create a new driver.

TPMC861_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 TPMC861_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 TPMC861 driver has changed and the makenode script isn't used.

2.6 FIFO Configuration

After installation of the TPMC861 Device Driver the trigger levels for both transmit and receive FIFO are set to their default values.

Default values are:

Receive FIFO	Transmit FIFO
96	32

The configuration of the FIFO trigger level is used for all TPMC861 devices in common.

To change the trigger levels edit the file *hal/tpmc861haldef.h,* change the following symbols to appropriate values and enter ***make install*** to create a new driver.

TPMC861_RX_TRG_DEF Define the trigger level for the receiver FIFO of a TPMC861 with XR16C864 controller.

Valid trigger levels are:
UART_TRG_1
UART_TRG_4
UART_TRG_8
UART_TRG_16
UART_TRG_32
UART_TRG_64
UART_TRG_96
UART_TRG_120
UART_TRG_128

TPMC861_TX_TRG_DEF Define the trigger level for the transmitter FIFO of a TPMC861 with XR16C864 controller.

Valid trigger levels are:
UART_TRG_1
UART_TRG_4
UART_TRG_8
UART_TRG_16
UART_TRG_32
UART_TRG_64
UART_TRG_96
UART_TRG_120
UART_TRG_128

Please refer to the User Manual of the XR16C864 controller to get more information how to customize suitable FIFO trigger level.

2.7 Configuration hints

After loading the devices the device configuration can be changed. Be sure if it makes sense to have echo enabled. It must be disabled for RS485 and it must never be enabled on both sides of a connection. By default the echo is enabled after loading the device. Configuration can be changed with the *stty* function.

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/ttySTPMC861_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

Error Code	Description
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

Error Code	Description
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>

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 *tpmc861.h*:

Value	Description
TPMC861_IOCQ_BIST	Start Built-In-Self-Test
TPMC861_IOCT_SPEED	Setup user defined baud rates
TPMC861_IOCQ_GET_SPEED	Returns the current configured baud rate

See below for more detailed information on each control code.

To use these TPMC861 specific control codes the header file *tpmc861.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

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

Other function dependent error codes will be described for each ioctl code separately. Note, the TPMC861 driver always returns standard Linux error codes.

SEE ALSO

ioctl man pages

3.3.1 TPMC861_IOCQ_BIST

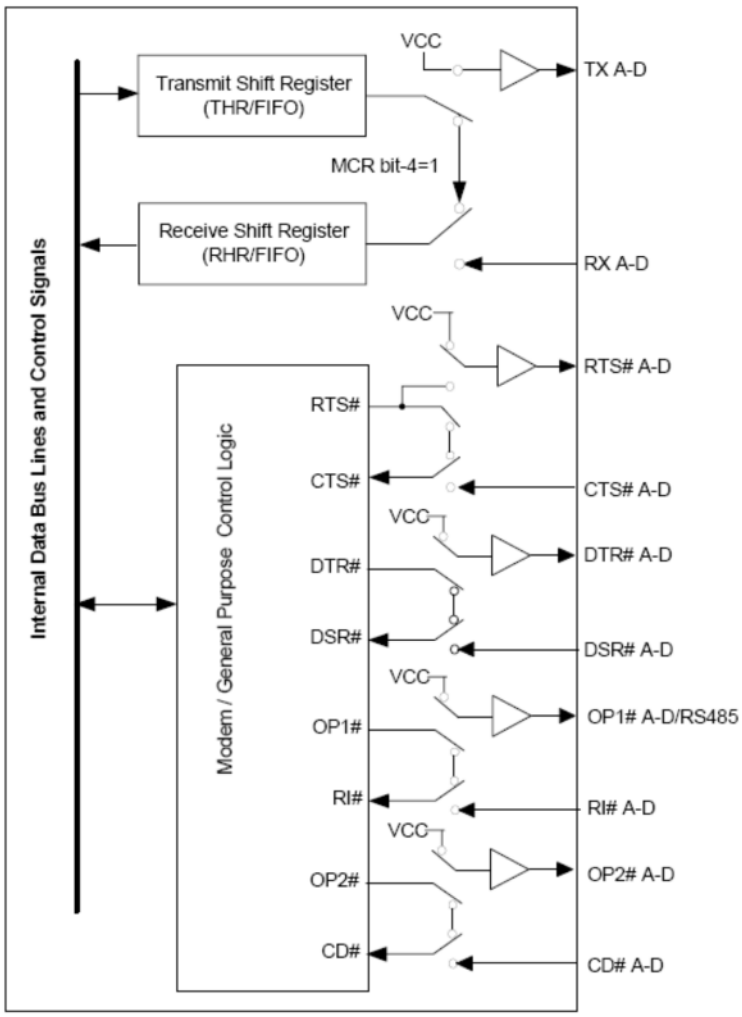
NAME

TPMC861_IOCQ_BIST – Start Built-In-Self-Test

DESCRIPTION

The TPMC861 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.



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 is set to 115200 BAUD for test buffer transmission. After the BIST completes the old channel settings are restored.

The last test verifies received data to assert data integrity.

EXAMPLE

```
#include “tpmc861.h”

int result;

/*
** Start Built-In Selftest, assumes an open device handle in “tty1”
*/
result = ioctl(tty1, TPMC861_IOCQ_BIST, NULL);

if (result < 0)
{
  printf("ERRNO %d - %s\n", errno, strerror(errno));
}
else if (result > 0)
{
  printf("Error during Built-In Selftest <0x%08X>!\n", result);

  if (result & TPMC861_ERTSCTS)
    printf("RTS/CTS line broken!\n");
  if (result & TPMC861_EDTRDSR)
    printf("DTR/DSR line broken!\n");
  if (result & TPMC861_ERI)
    printf("OP1/RI line broken!\n");
  if (result & TPMC861_ECD)
    printf("OP2/DCD line broken!\n");
  if (result & TPMC861_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.

Value	Description
TPMC861_ERTSCTS	If set RTS/CTS line broken.
TPMC861_EDTRDSR	If set DTR/DSR line broken.
TPMC861_ERI	If set OP1/RI line broken.
TPMC861_ECD	If set OP2/CD line broken.
TPMC861_EDATA	Data integrity test failed. No correct transmission possible.

ERRORS

Error Code	Description
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.

3.3.2 TPMC861_IOCT_SPEED

NAME

TPMC861_IOCT_SPEED – Setup user defined baud rates

DESCRIPTION

This ioctl function sets up a user defined baud rate. This allows using the TPMC861 device with every adjustable baud rate.

The new baud rate is passed by value by the parameter ***arg*** to the driver. The maximum baud rate limit is 460800 Baud.

If a user defined baud rate is set, standard tools (like *stty*) will return invalid information about the selected baud rate.

EXAMPLE

```
#include <tpmc861.h>

int result, tty1;

/* Setup 76800 Baud */
result = ioctl(tty1, TPMC861_IOCT_SPEED, 76800);

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

3.3.3 TPMC861_IOCQ_GET_SPEED

NAME

TPMC861_IOCQ_GET_SPEED – Read the actually configured baud rate

DESCRIPTION

This ioctl function returns the actually configured baud rate of the specified channel. This allows checking if a baud rate can be configured correctly or if it is substituted by the nearest configurable baud rate.

The current baud rate is returned in the integer argument the parameter ***arg*** points on.

EXAMPLE

```
#include <tpmc861.h>

int result, tty1, baudrate;

result = ioctl(tty1, TPMC861_IOCQ_GET_SPEED, &baudrate);

if (result < 0) {
   /* handle errors */
}
else {
   printf("Current Baudrate: %d\n", baudrate);
}
```

4 Device Driver Programming

The TPMC861 driver is loosely based 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.

The source files *tpmc861example.c, tpmc861setspeed.c and tpmc861bist.c* contain additional programming examples.

4.1 Setting up Baud Rates

The driver allows setting all baud rates supported by the channel. Not only standard baud rates are supported, also special baud rates are supported. The driver will always try to set the best matching baud rate.

There are two possibilities setting up baud rates:

The first is used to setup predefined baud rates, this is the standard way by using the termios structure (e.g. using stty).

The second way allows the selection of all baud rates the module can support. This way uses the ioctl function *TPMC861_IOCT_SPEED* (please refer to the description of the ioctl function).

5 Diagnostic

If the TPMC861 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 TPMC861 driver (see also the proc man pages).

```
# lspci -v
04:01.0 Multiport serial controller: TEWS Technologies GmbH TPMC861 4-
Channel Isolated Serial Interface RS422/RS485 (rev 0a)
        Subsystem: TEWS Technologies GmbH Device 000a
        Flags: medium devsel, IRQ 16
        Memory at feb9fc00 (32-bit, non-prefetchable) [size=128]
        I/O ports at e880 [size=128]
        Memory at feb9f800 (32-bit, non-prefetchable) [size=64]
        Kernel driver in use: TEWS TECHNOLOGIES - TPMC861HAL Driver
        Kernel modules: tpmc861haldrv


# lsmod | grep tpmc861
tpmc861serialdrv      325464  0
tpmc861haldrv          18023  1 tpmc861serialdrv


# ls /dev | grep 861
ttySTPMC861_0
ttySTPMC861_1
ttySTPMC861_2
ttySTPMC861_3


# cat /proc/tty/driver/tpmc861serial
serinfo:1.0 driver revision:
0: uart:XR16C864 mmio:0xFEB9F800 irq:16 tx:11024 rx:1024
1: uart:XR16C864 mmio:0xFEB9F808 irq:16 tx:1024 rx:1024
2: uart:XR16C864 mmio:0xFEB9F810 irq:16 tx:2048 rx:2048
3: uart:XR16C864 mmio:0xFEB9F818 irq:16 tx:1024 rx:11024
```