The Embedded I/O Company

TEWS TECHNOLOGIES

TPMC866-SW-82

Linux Device Driver

8 Channel Serial PMC

Version 2.1.x

User Manual

Issue 2.1.0

April 2012

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

TPMC866-SW-82

Linux Device Driver

8 Channel Serial PMC

Supported Modules:
TPMC866

Issue	Description	Date
1.0	First Issue	February 10, 2000
1.1	Support for Kernel 2.4	August 21, 2001
1.2	Full Modem Support for TPMC866-10 (1st+2nd Channel) Modification for Kernel 2.4.18	June 19, 2003
2.0.0	Support for Kernel 2.6, DEVFS and UDEV Build-In-Self-Test function (TPMC866_IOCQ_BIST) selectable feature: full modem support	April 28, 2006
2.0.1	File list modified, general revision, New Address TEWS LLC	September 19, 2008
2.0.2	New top level Makefile, diagnostic info for kernel 2.6 added and archive file list corrected, address TEWS LLC removed	August 9, 2010
2.1.0	New file structure, support of Kernel 3.x.x	April 12, 2012

Table of Contents

1 Introduction

The TPMC866 Linux device driver is a full-duplex serial driver which allows the operation of a TPMC866 serial PMC on Linux operating systems.

The TPMC866 device driver is based on the standard Linux serial device driver and supports all standard terminal functions (TERMIOS).

The TPMC866 device driver includes the following features:

- Extended baud rates up to 460800 Baud
- Each channel uses the transmit and receive FIFO (FIFO size depends on the module)
- 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
- Designed as Linux kernel module with dynamic loading
- Supports shared IRQ's
- Built on new style PCI driver layout
- Creates TTY devices (ttySTPMC866_x) 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

Selectable features (see chapter Installation)

- Full-Modem support for 1st and 2nd channel
- Creates a dialout device cuaTPMC866 (Kernel 2.4.x) with dynamically allocated or fixed major device numbers.

The TPMC866-SW-82 device driver supports the modules listed below:

TPMC866-10/-11	8 Channel Serial Interface (ST16C654)	(PMC)
TPMC866-12	8 Channel Serial Interface (XR16C864)	(PMC)

To get more information about the features and use of TPMC866 device it is recommended to read the manuals listed below.

TPMC866 User manual (TPMC866-10/-11)
TPMC866 Engineering Manual (TPMC866-10/-11)
ST16C654 UART Hardware Manual (TPMC866-10/-11)
TPMC866-12 User manual (TPMC866-12)
TPMC866-12 Engineering Manual (TPMC866-12)
XR16C864 UART Hardware Manual (TPMC866-12)

2 Installation

The directory TPMC866-SW-82 on the distribution media contains the following files:

TPMC866-SW-82-2.1.0.pdf	This manual in PDF format
TPMC866-SW-82-SRC.tar.gz	GZIP compressed archive with driver source code
Release.txt	Release information
ChangeLog.txt	Release history

The GZIP compressed archive TPMC866-SW-82-SRC.tar.gz contains the following files and directories:

example/Makefile	Example application makefile
example/tpmc866example.c	Send and receive example application
example/tpmc866setspeed.c	Speed configuration example application
example/tpmc866bist.c	Example for using Built-In-Self-Test
hal/	Hardware abstraction layer driver needed for all kernel versions
hal/Makefile	HAL driver makefile
hal/tpmc866hal.c	HAL driver source file
hal/tpmc866haldef.h	HAL driver private header file
serial/	UART driver directory
serial/2.4.x/Makefile	Serial driver makefile (kernel 2.4.x)
serial/2.4.x/tpmc866serial.c	Serial driver source file (kernel 2.4.x)
serial/2.4.x/tpmc866serialdef.h	Serial driver private header file (kernel 2.4.x)
serial/Makefile	Serial driver makefile (Kernel 2.6.x+)
serial/tpmc866serial.c	Serial driver source file (Kernel 2.6.x+)
serial/tpmc866serialdef.h	Serial driver private header file (Kernel 2.6.x+)
serial/makenode	Shell script to create devices nodes without DEVFS
serial/makenodeFM24	Alternative shell script to create devices nodes without DEVFS
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
tpmc866def.h	Driver private header file
tpmc866.h	User application header file
Makefile	Top-level Makefile

In order to perform an installation, extract all files of the archive TPMC866-SW-82-SRC.tar.gz to the desired target directory. The command 'tar -xzvf TPMC866-SW-82-SRC.tar.gz' will extract the files into the local directory.

- Login as *root* and change to the target directory
- Copy tpmc866.h to */usr/include*

2.1 Build and install the Device Driver

- Login as *root*
- Change to the tpmc866 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 tpmc866 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 tpmc866serialdrv

- 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 dynamic 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

If the selectable feature TPMC866_ENA_FULLMODEM has been enabled for a system running 2.4.x kernel, *makenodeFM24* should be used for device node creation instead of *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/ttySTPMC866_0, the second channel with device node /dev/ttySTPMC866_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 tpmc866serialdrv

If your kernel has enabled a dynamic device file system (devfs, udev, ...), all /dev/ttySTPMC866_* 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 "*tpmc866serialdrv: 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 dynamic device file system installed.

The released TPMC866 driver uses 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>/tpmc866serial.c, change the following symbols to appropriate values and enter *make install* to create a new driver.

TPMC866_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.
TPMC866_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 TPMC866_TTY_MAJOR       122
#define TPMC866_CUA_MAJOR       123
```

The definition of the major number for the CUA driver is only used if the selectable feature for full modem support *TPMC866_ENA_FULLMODEM* is enabled

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 TPMC866 driver has changed and the makenode script isn't used.

2.6 FIFO Configuration

After installation of the TPMC866 Device Driver the trigger level for transmit and receive FIFO are set to their default values.

Default values are:

Receive FIFO	Transmit FIFO	Module Type
56	16	TPMC866-10/-11
96	32	TPMC866-12

The configuration of the FIFO trigger level is used for all TPMC866 devices in common.

To change the FIFO trigger levels edit the file *hal/tpmc866haldef.h,* change the following symbols to appropriate values and enter ***make install*** to create a new driver.

TPMC866_10_RX_TRG_DEF
Define the trigger level for the receiver FIFO of a TPMC866 with ST16C654 controller (TPMC866-10/-11):

Valid trigger levels are:
UART_FCR_R_TRIGGER_60
UART_FCR_R_TRIGGER_56 (set by default)
UART_FCR_R_TRIGGER_16
UART_FCR_R_TRIGGER_8

TPMC866_10_TX_TRG_DEF
Define the trigger level for the transmitter FIFO of a TPMC866 with ST16C654 controller (TPMC866-10/-11):

Valid trigger levels are:
UART_FCR_T_TRIGGER_56
UART_FCR_T_TRIGGER_32
UART_FCR_T_TRIGGER_16 (set by default)
UART_FCR_T_TRIGGER_8

TPMC866_12_RX_TRG_DEF
Define the trigger level for the receiver FIFO of a TPMC866 with XR16C864 controller (TPMC866-12). Valid values are 1 to 128, the default value is 96.

TPMC866_12_TX_TRG_DEF
Define the trigger level for the transmitter FIFO of a TPMC866 with XR16C864 controller (TPMC866-12). Valid values are 1 to 128, the default value is 16.

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

2.7 Selectable Features

Full modem support can be enabled by defining the symbol *TPMC866_ENA_FULLMODEM* in "tpmc866def.h".

Enabling this feature may lead to extra interrupts on serial channels that do not support full modem lines. Extra interrupts will lead to loss of system performance. Therefore we recommend not using full modem support until it is needed.

2.8 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 shall 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 Driver Programming

The TPMC866 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.

3.1 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 *tpmc866.h*:

Value	Meaning
TPMC866_IOCQ_BIST	Start Built-In-Self-Test

See below for more detailed information on each control code.

To use these TPMC866 specific control codes the header file *tpmc866.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 TPMC866 driver always returns standard Linux error codes.

SEE ALSO

ioctl man pages

3.1.1 TPMC866_IOCQ_BIST

NAME

TPMC866_IOCQ_BIST – Start Built-In-Self-Test

DESCRIPTION

The TPMC866 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.

Communication parameters like baud rate, data length, etc. are configured during the BIST and restored after the BIST is completed.

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

The line test contains a test of all modem lines pairs (RTS and CTS, DTR and DSR, OP1 and RI, 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 the same channel during internal loopback. If not, there is an interrupt error. The buffer size is 1024 byte.

The last test verifies received data to assert data integrity.

This function tests all internal I/O lines of the controller, even if they are not used for interfacing.

EXAMPLE

```
#include <tpmc866.h>

/* Start Built-In Selftest, */
result = ioctl(tty1, TPMC866_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 & TPMC866ERTSCTS)
    printf("RTS/CTS line broken!\n");
  if (result & TPMC866_EDTRDSR)
    printf("DTR/DSR line broken!\n");
  if (result & TPMC866_ERI)
    printf("OP1/RI line broken!\n");
  if (result & TPMC866_ECD)
    printf("OP2/DCD line broken!\n");
  if (result & TPMC866_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
TPMC866_ERTSCTS	If set RTS/CTS line broken.
TPMC866_EDTRDSR	If set DTR/DSR line broken.
TPMC866_ERI	If set OP1/RI line broken.
TPMC866_ECD	If set OP2/CD line broken.
TPMC866_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.

4 Diagnostic

If the TPMC866 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 TPMC866 driver (see also the proc man pages).

```
# lspci –v
04:01.0 Multiport serial controller: PLX Technology, Inc. PCI <-> IOBus
Bridge (rev 01)
        Subsystem: TEWS Technologies GmbH TPMC866 8 Channel Serial Card
        Flags: medium devsel, IRQ 16
        Memory at feb9fc00 (32-bit, non-prefetchable) [size=128]
        I/O ports at e880 [size=128]
        I/O ports at e800 [size=128]
        Kernel driver in use: TEWS TECHNOLOGIES - TPMC866HAL Driver
        Kernel modules: tpmc866haldrv, hisax

# lsmod | grep tpmc866
tpmc866serialdrv      620461  0
tpmc866haldrv          30356  1 tpmc866serialdrv

# ls /dev | grep 866
ttySTPMC866_0
ttySTPMC866_1
ttySTPMC866_2
ttySTPMC866_3
ttySTPMC866_4
ttySTPMC866_5
ttySTPMC866_6
ttySTPMC866_7

# cat /proc/tty/driver/tpmc866serial
serinfo:1.0 driver revision:
0: uart:ST16C654 port:0000E800 irq:16 tx:0 rx:0
1: uart:ST16C654 port:0000E808 irq:16 tx:0 rx:0
2: uart:ST16C654 port:0000E810 irq:16 tx:0 rx:0 DSR|CD|RI
3: uart:ST16C654 port:0000E818 irq:16 tx:0 rx:0 DSR|CD|RI
4: uart:ST16C654 port:0000E820 irq:16 tx:0 rx:0 DSR|CD|RI
5: uart:ST16C654 port:0000E828 irq:16 tx:0 rx:0 DSR|CD|RI
6: uart:ST16C654 port:0000E830 irq:16 tx:0 rx:0 DSR|CD|RI
7: uart:ST16C654 port:0000E838 irq:16 tx:0 rx:0 DSR|CD|RI
```