TDRV008-SW-82

Linux Device Driver

3 x 16 bit I/O Ports with 512 Word FIFO and Handshake

Version 1.0.x

User Manual

Issue 1.0.1

January 2021

TDRV008-SW-82

Linux Device Driver

3 x 16 bit I/O Ports with 512 Word FIFO and Handshake

Supported Modules:

      TPMC682

Issue	Description	Date
1.0.0	First Issue	May 26, 2008
1.0.1	Correction of supported modules, Filelist modified	January 27, 2021

Table of Contents

1 <u>Introduction</u>

The TDRV008-SW-82 Linux device driver allows the operation of the TDRV008 compatible devices conforming to the Linux I/O system specification. This includes a device-independent basic I/O interface with *open()*, *close(),*and *ioctl()* functions.

Special I/O operation that do not fit to the standard I/O calls will be performed by calling the *ioctl()* function with a specific function code and an optional function dependent argument.

The TDRV008-SW-82 device driver supports the following features:

➢ write to the 8-bit GPO port 5
➢ read from the 8-bit GPI port 4
➢ configure ports (direction, mode and hardware timeout)
➢ buffered read and write of the 16-bit ports (0, 1 & 2) in pulsed or interlocked handshake mode
➢ hardware FIFO flush

<u>The TDRV008-SW-82 supports the modules listed below:</u>

TPMC682	3 x 16 bit I/O Ports with 512 Word FIFO and Handshake	PMC

> **In this document all supported modules and devices will be called TDRV008. Specials for a certain device will be advised.**

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

TPMC682 (or compatible) User Manual

2 <u>Installation</u>

Following files are located on the distribution media:

TDRV008-SW-82-1.0.1.pdf	This manual in PDF format
TDRV008-SW-82-SRC.tar.gz	GZIP compressed archive with driver source code
ChangeLog.txt	Release history
Release.txt	Information about the Device Driver Release

The GZIP compressed archive TDRV008-SW-82-SRC.tar.gz contains the following files and directories:

tdrv008.c	Driver source code
tdrv008def.h	Driver include file
tdrv008.h	Driver include file for application program
Makefile	Device driver make file
makenode	Script for device node creation
include/config.h	Driver independent configuration header file
include/tpmodule.c	Driver independent library
include/tpmodule.h	Driver independent library header file
example/tdrv008exa.c	Example application
example/Makefile	Example application makefile
COPYING	Copy of the GNU Public License (GPL)

In order to perform an installation, extract all files of the archive TDRV008-SW-82-SRC.tar.gz to the desired target directory. The command 'tar -xzvf TDRV008-SW-82-SRC.tar.gz' will extract the files into the local directory.

- Login as *root* and change to the target directory

- Copy tdrv008.h to */usr/include*

2.1 Build and install the device driver

- Login as *root*

- Change to the target directory

- To create and install the 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 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 tdrv008drv

- After the first build or if you are using dynamic major device allocation it is necessary to create new device nodes on the file system. Please execute the script file *makenode* to do this. If your kernel has enabled a device file system (devfs or sysfs with udev) then you have to 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 TDRV008 device found. The first TDRV008 device can be accessed with device node /dev/tdrv008_0, the second module with device node /dev/tdrv008_1 and so on.

The assignment of device nodes to physical TDRV008 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 tdrv008drv

If your kernel has enabled devfs or sysfs (udev), all /dev/tdrv008_x 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 "*tdrv008drv: 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 TDRV008 driver uses dynamic allocation of major device numbers per default. If this isn't suitable for the application it is possible to define a major number for the driver.

To change the major number, edit the file tdrv008def.h, change the following symbol to appropriate value and enter `make install` to create a new driver.

TDRV008_MAJOR   Valid numbers are in range between 0 and 255. A value of 0 means dynamic number allocation.

Example:

```
#define TDRV008_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.

3 Device Input/Output functions

This chapter describes the interface to the device driver I/O system.

3.1 open

NAME

open()   opens a file descriptor.

SYNOPSIS

```
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. 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

```
#include <fcntl.h>

int  fd;

…

fd = open("/dev/tdrv008_0", O_RDWR);
if (fd == -1)
{
   /* handle error condition */
}
```

RETURNS

The normal return value from open is a non-negative integer file descriptor. In case of an error, a value of −1 is returned. The global variable *errno* contains the detailed error code.

ERRORS

Error Code	Description
E_NODEV	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()   closes a file descriptor.

SYNOPSIS

```
int close
(
    int     filedes
)
```

DESCRIPTION

The **close** function closes the file descriptor *filedes*.

EXAMPLE

```
#include <unistd.h>

int fd;

…

if (close(fd) != 0)
{
   /* handle close error conditions */
}
```

RETURNS

The normal return value from close is 0. In case of an error, a value of −1 is returned. The global variable *errno* contains the detailed error code.

ERRORS

Error Code	Description
E_NODEV	The requested file descriptor 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

```
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 *tdrv008.h*:

Value	Meaning
TDRV008_IOCX_READ	Buffered read from a 16-bit handshake port
TDRV008_IOCX_WRITE	Buffered write to a 16-bit handshake port
TDRV008_IOCG_GETPORT	Get the state of the 8-bit GPI port #4
TDRV008_IOCS_SETPORT	Set the state of the 8-bit GPO port #5
TDRV008_IOCS_CONFPORT	Configure handshake port
TDRV008_IOC_FLUSHPORTS	Flush FIFOs of all handshake ports

See below for more detailed information on each control code.

> **To use these TDRV008 specific control codes the header file *tdrv008.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 TDRV008 device driver always returns standard Linux error codes.

SEE ALSO

ioctl man pages

3.3.1 TDRV008_IOCX_READ

NAME

TDRV008_IOCX_READ   Buffered read from a 16-bit handshake port

DESCRIPTION

This ioctl function reads 16-bit values from the FIFO of a given port. If data isn't available when calling this function a timeout is used to implement a blocking read. A pointer to the caller's data buffer (*TDRV008_RW_BUFFER*) is passed to the device driver by the argument *argp*.

```
typedef struct
{
        int                     portNo;
        unsigned long           flags;
        int                     timeout;
        int                     bufferSize;
        int                     validWords;
        unsigned short          data[TDRV008_FIFOSIZE];

} TDRV008_RW_BUFFER, *PTDRV008_RW_BUFFER;
```

portNo

> This parameter holds the port number to read from. Valid values are 0, 1 and 2.

flags

> This parameter decides about non-blocking and blocking read requests. For non-blocking operation set *flags* to TDRV008_F_RW_NOWAIT. To initialize a blocking read request set *flags* to zero.

timeout

> This parameter defines the read timeout with a given resolution of one system tick.

bufferSize

> This parameter defines the maximum count of 16 bit words to read.

validWords

> This in and out parameter holds the count of data words actually read. After read requests completion maybe not all data words given by *bufferSize* were received in the given time. For this purpose *validWords* is used for the read request result. If it is set to a value greater than 0 when starting the read request and blocking read is used then *validWords* is meant as an offset to the first element of the *data* buffer. So successive read requests that are partially completed can use the same I/O buffer until final read request completion.

data

> This field parameter is used as data storage. It has a fixed size of TDRV008_FIFOSIZE (512) words to match the hardware FIFO size.

EXAMPLE

```
#include "tdrv008.h"

int                 fd;
int                 result, i;
TDRV008_RW_BUFFER   rwBuf;

rwBuf.portNo      = 1;          // read from port 1
rwBuf.flags       = 0;          // blocking read
rwBuf.timeout     = 5;          // wait at least 5 seconds
rwBuf.bufferSize  = 177;        // we want to receive 177 words
rwBuf.validWords  = 0;          // no data received yet
/* rwBuf.data[] will be filled with incoming data words */

result = ioctl(fd, TDRV008_IOCX_READ, &rwBuf);

if (result < 0) {
  /* handle ioctl error */
} else {
  // Process data, rwBuf.validWords is the real read result
  for (i = 0; i < rwBuf.validWords; i++)
  {
    printf( "@0x%04X: 0x%02X", i, rwBuf.data[i] );
  }
  …
}
```

ERRORS

Error Code	Description
EINVAL	Invalid arguments in structure.
EACCES	Invalid port direction: Port is configured to output.
EBUSY	The port is already busy.
ETIME	Timeout occurred.
EFAULT	Error while copying data to or from user space.

Other returned error codes are system error conditions.

3.3.2 TDRV008_IOCX_WRITE

NAME

TDRV008_IOCX_WRITE   Buffered write to a 16-bit handshake port

DESCRIPTION

This ioctl function writes 16-bit values to the specified buffered output port. A pointer to the caller's data buffer (*TDRV008_RW_BUFFER*) is passed to the device driver by the argument *argp.*

typedef struct
{

int	portNo;
unsigned long	flags;
int	timeout;
int	bufferSize;
int	validWords;
unsigned short	data[TDRV008_FIFOSIZE];

} TDRV008_RW_BUFFER, *PTDRV008_RW_BUFFER;

portNo

> This parameter holds the port number of the handshake port to write to. Valid values are 0, 1 and 2.

flags

> This parameter is not used for this function.

timeout

> This parameter defines the write timeout with a given resolution of one system tick.

bufferSize

> This parameter defines the count of words to write to the hardware FIFO of the certain handshake port.

validWords

> This in and out parameter holds the count of data words actually written. In the case of a full FIFO the write process can't send more data words to the certain port and will block for *timeout* seconds. For this purpose *validWords* is used for the write request result. If it is set to a value greater than 0 when starting the write request then *validWords* is meant as an offset to the first element of the *data* buffer. Successive write requests that are partially completed can use the same I/O buffer until final write request completion.

data

> This field parameter is used as data source during the handshake transmission. It has a fixed maximum size of TDRV008_FIFOSIZE (512) words to match the hardware FIFO size.

EXAMPLE

```
#include "tdrv008.h"


int                 fd;
int                 result, i;
TDRV008_RW_BUFFER   rwBuf;


rwBuf.portNo        = 0;            // write to port 0
rwBuf.timeout       = 200;          // wait at least 200 ticks
rwBuf.bufferSize    = 411;          // we want to send 411 words
rwBuf.validWords    = 0;            // start with element 0


for (i = 0; i < rwBuf.bufferSize; i++)
{
   rwBuf.data[i] = ...;             // user data
}


result = ioctl(fd, TDRV008_IOCX_WRITE, &rwBuf);


if (result < 0) {
   /* handle ioctl error */
} else {
   // Process data, rwBuf.validWords is the real write result
   …
}
```

ERRORS

Error Codes	Description
EINVAL	Invalid arguments in structure.
EACCES	Invalid port direction: Port is configured to input.
EBUSY	The port is already busy.
ETIME	Timeout occurred.
EFAULT	Error while copying data to or from user space.

Other returned error codes are system error conditions.

3.3.3 TDRV008_IOCG_GETPORT

NAME

TDRV008_IOCG_GETPORT   Get the state of the 8-bit GPI port #4

DESCRIPTION

This TDRV008 control function reads the state of the free input lines of the 8 bit general purpose port 4. Only the upper 5 bit of the value are valid the lower 3 bits will always be set to 0. A pointer to an *unsigned char* value is passed to the driver by the argument *argp*.

EXAMPLE

```
#include "tdrv008.h"

int           fd;
int           result;
unsigned char PortState;

result = ioctl(fd, TDRV008_IOCG_GETPORT, &PortState);

if (result < 0) {
    /* handle ioctl error */
} else {
    printf( "Port4 (bit7..3): %02Xh\n", PortState );
}
```

ERRORS

Error Code	Description
EFAULT	Error while copying data to user space.

Other returned error codes are system error conditions.

3.3.4 TDRV008_IOCS_SETPORT

NAME

TDRV008_IOCS_SETPORT   Set the state of the 8-bit GPO port #5

DESCRIPTION

This TDRV008 control function sets the state of the free output lines of the general purpose port 5. Only the upper 5 bit of the value are valid the lower 3 bits are ignored. A pointer to an *unsigned char* value is passed to the driver by the argument *argp*.

EXAMPLE

```
#include "tdrv008.h"

int          fd;
int          result;
unsigned char PortState;

PortState = 0x42;     // bits 0..2 are ignored -> so 0x40 will be written

result = ioctl(fd, TDRV008_IOCS_SETPORT, &PortState);

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

ERRORS

Error Code	Description
EFAULT	Error while copying data to user space.

Other returned error codes are system error conditions.

3.3.5 TDRV008_IOCS_CONFPORT

NAME

TDRV008_IOCS_CONFPORT   Configure handshake port

DESCRIPTION

This TDRV008 control function configures a specified handshake port. A pointer to the caller's data buffer (*TDRV008_CONF_BUFFER*) is passed to the device driver by the argument *argp*.

typedef struct
{
        int   portNo;
        unsigned long   flags;
        int   enaOutput;
        unsigned short   fifoTimeout;
        unsigned short   fifoThreshold;

} TDRV008_ CONF_BUFFER, *PTDRV008_ CONF_BUFFER;

portNo

    This parameter specifies the handshake port. Valid values are 0, 1 and 2.

flags

    This parameter specifies the output handshake mode. (Refer to the User Manual of your module for a detailed description of the output handshake modes). Following values are valid.

Value	Description
TDRV008_F_CONF_HOUT_HSNONE	No output handshake
TDRV008_F_CONF_HOUT_HSINTERLOCKED	Interlocked output handshake
TDRV008_F_CONF_HOUT_HSPULSED	Pulsed output handshake

enaOutput

    This parameter defines the direction of the port. If this parameter is set *TRUE* (1) the port will be configured as an output port, if it is specified *FALSE* (0) the port will be configured as input.

fifoTimeout

    This parameter specifies the hardware FIFO timeout value. The value will be directly written to the module (Register TCPRx - refer to the User Manual of your module for more information). This value is only used for input ports.

fifoThreshold

    This parameter specifies the FIFO threshold value. This value will be directly written to the module (Register FIFO_FTRx - refer to the User Manual of your module for more information). Valid values 1 to 512.

EXAMPLE

```
#include "tdrv008.h"

int                 fd;
int                 result;
TDRV008_CONF_BUFFER confBuf;

/* Setup handshake port 0 */
/* - output */
/* - interlocked output handshake */
/* - threshold: 256 */
confBuf.portNo       = 0;
confBuf.flags        = TDRV008_F_CONF_HOUT_HSINTERLOCKED;
confBuf.enaOutput    = TRUE;
confBuf.fifoTimeout  = 0; /* not used */
confBuf.fifoThreshold = 256;

result = ioctl(fd, TDRV008_IOCS_CONFPORT, &confBuf);

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

ERRORS

Error Code	Description
EFAULT	Error while copying data to user space.
EINVAL	Invalid parameter specified.
EBUSY	The port is busy with data transfer.

Other returned error codes are system error conditions.

3.3.6 TDRV008_IOC_FLUSHPORTS

NAME

TDRV008_IOC_FLUSHPORTS   Flush FIFOs of all handshake ports

DESCRIPTION

This TDRV008 control function flushes the FIFOs of all handshake ports (0, 1, and 2). This may be useful after configuration. No additional parameter is used for this function, so the optional argument can be omitted.

EXAMPLE

```
#include "tdrv008.h"

int fd;
int result;

result = ioctl(fd, TDRV008_IOC_FLUSHPORTS);

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

ERRORS

There are no function specific errors. Returned error codes are system error conditions.

4 <u>Diagnostic</u>

If the TDRV008 does not work properly it is helpful to get some status information from the driver respective kernel. To get debug output from the driver enable the following symbols in "tdrv008.c" by replacing "#undef" with "#define", recompile and reinstall the driver:

```
#define DEBUG_TDRV008
```

The Linux */proc* file system provides additional information about kernel, resources, drivers, devices and so on. The following screen dumps display information of a correct running TDRV008 driver (see also the proc man pages).

```
# cat /proc/devices
Character devices:
  1 mem
  2 pty
  …
136 pts
162 raw
254 tdrv008drv
  …


# cat /proc/iomem
00000000-00000fff : Reserved
00001000-0009fbff : System RAM
…
  feb00000-febfffff : PCI Bus 0000:04
    feb9f400-feb9f40f : 0000:04:01.0
      feb9f400-feb9f40f : TDRV008
    feb9f800-feb9f83f : 0000:04:01.0
      feb9f800-feb9f83f : TDRV008
    feb9fc00-feb9fc7f : 0000:04:01.0
…
100000000-11ffffff : System RAM
```