The Embedded I/O Company

TEWS TECHNOLOGIES

TDRV007-SW-42

VxWorks Device Driver

ARCNET

Version 1.0.x

User Manual

Issue 1.0.0

March 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

TDRV007-SW-42

ARCNET

VxWorks Device Driver

Supported Modules:
TPMC815
THP815

Issue	Description	Date
1.0.0	First Issue	March 1, 2006

Table of Content

1 Introduction

The TDRV007-SW-42 VxWorks device driver software allows the operation of the TPMC815 product family conforming to the VxWorks 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.

This driver invokes a mutual exclusion and binary semaphore mechanism to prevent simultaneous requests by multiple tasks from interfering with each other.

The TDRV007-SW-42 device driver supports the following features:

- Configure ARCNET node and set node online
- Remove node from ARCNET (set offline)
- Send messages
- Receive messages
- Read ARCNET map
- Get diagnostic information (reconfiguration cycles)

The TDRV007-SW-42 supports the modules listed below:

TPMC815	1 Channel ARCNET	PMC
THP815	1 Channel ARCNET	PC/104-Plus

To get more information about the features and use of TDRV007 devices it is recommended to read the manuals listed below.

User manual of supported module

Engineering Manual of supported module

2 Installation

Following files are located on the distribution media:

Directory path '.\TDRV007-SW-42\':

tdrv007drv.c	TDRV007 device driver source
tdrv007def.h	TDRV007 driver include file
tdrv007.h	TDRV007 include file for driver and application
tdrv007pci.c	TDRV007 PCI MMU mapping for Intel x86 based targets
tdrv007exa.c	Example application
tdhal.h	Hardware dependent interface functions and definitions
arcdef.h	Definitions for ARCNET Controller Hardware
TDRV007-SW-42-1.0.0.pdf	PDF copy of this manual
Release.txt	Release information

2.1 Include device driver in Tornado IDE project

For Including the TDRV007-SW-42 device driver into a Tornado IDE project follow the steps below:

(1) Copy the files from the distribution media into a subdirectory in your project path.
(For example: ./TDRV007)

(2) Add the device driver's C-files to your project.
Make a right click to your project in the 'Workspace' window and use the 'Add Files ...' topic.
A file select box appears, and the driver files can be selected.

(3) Now the driver is included in the project and will be built with the project.

For a more detailed description of the project facility please refer to your Tornado User's Guide.

2.2 Special installation for Intel x86 based targets

The TDRV007 device driver is fully adapted for Intel x86 based targets. This is done by conditional compilation directives inside the source code and controlled by the VxWorks global defined macro **CPU_FAMILY**. If the content of this macro is equal to *I80X86* special Intel x86 conforming code and function calls will be included.

The second problem for Intel x86 based platforms can't be solved by conditional compilation directives. Due to the fact that some Intel x86 BSP's doesn't map PCI memory spaces of devices which are not used by the BSP, the required device memory spaces can't be accessed.

To solve this problem a MMU mapping entry has to be added for the required TDRV007 PCI memory spaces prior the MMU initialization (*usrMmuInit()*) is done.

The C source file **tdrv007pci.c** contains the function *tdrv007PciInit()*. This routine finds out all TDRV007 devices and adds MMU mapping entries for all used PCI memory spaces. Please insert a call to this function after the PCI initialization is done and prior to MMU initialization (*usrMmuInit()*).

The right place to call the function *tdrv007PciInit()* is at the end of the function *sysHwInit()* in **sysLib.c** (it can be opened from the project *Files* window).

Be sure that the function is called prior to MMU initialization otherwise the TDRV007 PCI spaces remains unmapped and an access fault occurs during driver initialization.

Please insert the following call at a suitable place in **sysLib.c**:

```
tdrv007PciInit();
```

Modifying the sysLib.c file will change the sysLib.c in the BSP path. Remember this for future projects and recompilations.

2.3 System resource requirement

The table gives an overview over the system resources that will be needed by the driver.

Resource	Driver requirement	Devices requirement
Memory	< 1 KB	< 5 KB
Semaphores	0	>= 4 (*1)

(*1) This value depends on the number of I/O request at the same time.

Memory and Stack usage may differ from system to system, depending on the used compiler and its setup.

The following formula shows the way to calculate the common requirements of the driver and devices.

*<total requirement> = <driver requirement> + (<number of devices> * <device requirement>)*

The maximum usage of some resources is limited by adjustable parameters. If the application and driver exceed these limits, increase the according values in your project.

3 I/O system functions

This chapter describes the driver-level interface to the I/O system. The purpose of these functions is to install the driver in the I/O system, add and initialize devices.

3.1 tdrv007Drv()

NAME

tdrv007Drv() - installs the TDRV007 driver in the I/O system

SYNOPSIS

#include “tdrv007.h”

STATUS tdrv007Drv(void)

DESCRIPTION

This function searches for devices on the PCI bus and installs the TDRV007 driver in the I/O system. The function makes initial initializations of the devices.

The call of this function is the first thing the user has to do before adding any device to the system or performing any I/O request.

EXAMPLE

```
#include  "tdrv007.h”

…

/*-------------------
  Initialize Driver
  -------------------*/
status = tdrv007Drv();
if (status == ERROR)
{
     /* Error handling */
}

…
```

RETURNS

OK, or ERROR if the function fails an error code will be stored in *errno*.

ERROR CODES

The error codes are stored in *errno* and can be read with the function *errnoGet()*.

Error code	Description
S_tdrv007Drv_NOMEM	Driver can not allocate memory for devices control block
S_tdrv007Drv_NXIO	Driver has not found a supported module

SEE ALSO

VxWorks Programmer's Guide: I/O System

3.2 tdrv007DevCreate()

NAME

tdrv007DevCreate() – Add a TDRV007 device to the VxWorks system

SYNOPSIS

#include “tdrv007.h”

```
STATUS tdrv007DevCreate
(
    char        *name,
    int         devIdx,
    int         funcType,
    void        *pParam
)
```

DESCRIPTION

This routine adds the selected device to the VxWorks system. The device hardware will be setup and prepared for use.

This function must be called before performing any I/O request to this device.

PARAMETER

name

This string specifies the name of the device that will be used to identify the device, for example for *open()* calls.

devIdx

This index number specifies the device to add to the system. The index number depends on the search priority of the modules. The modules will be searched in the following order:

- TPMC815-xx
- THP815-xx

If modules of the same type are installed the channel numbers will be assigned in the order the VxWorks *pciFindDevice()* function will find the devices.

Example: (A system with 1 TPMC815-xx and 2 THP815-xx) will assign the following device indexes:

Module	Device Index
TPMC815-xx	0
THP815-xx (1st)	1
THP815-xx (2nd)	2

funcType

This parameter is unused and should be set to *0*.

pParam

This parameter is unused and should be set to *NULL*.

EXAMPLE

```
#include  "tdrv007.h"

…

STATUS             result;

…

/*-----------------------------------------------------
  Create the device "/tdrv007/0" for the first device
  -----------------------------------------------------*/
result = tdrv007DevCreate(   "/tdrv007/0",
                             0,
                             0,
                             NULL);
if (result == OK)
{
     /* Device successfully created */
}
else
{
     /* Error occurred when creating the device */
}

…
```

RETURNS

OK, or ERROR if the function fails an error code will be stored in *errno*.

ERROR CODES

The error codes are stored in *errno* and can be read with the function *errnoGet()*.

Error code	Description
S_tdrv007Drv_NODRV	Driver has not been initialized
S_tdrv007Drv_NODEV	The specified device is not installed
S_tdrv007Drv_EXISTS	The device has been created before

SEE ALSO

VxWorks Programmer's Guide: I/O System

3.3 tdrv007PciInit()

NAME

tdrv007PciInit() – Generic PCI device initialization

SYNOPSIS

void tdrv007PciInit()

DESCRIPTION

This function is required only for Intel x86 VxWorks platforms. The purpose is to setup the MMU mapping for all required TPMC007 PCI spaces (base address register) and to enable the TDRV007 device for access.

The global variable *tdrv007Status* obtains the result of the device initialization and can be polled later by the application before the driver will be installed.

Value	Meaning
> 0	Initialization successful completed. The value of tdrv007Status is equal to the number of mapped PCI spaces
0	No TDRV007 device found
< 0	Initialization failed. The value of (tdrv007Status & 0xFF) is equal to the number of mapped spaces until the error occurs. Possible cause: Too few entries for dynamic mappings in sysPhysMemDesc[]. Remedy: Add dummy entries as necessary (syslib.c).

EXAMPLE

```
extern void tdrv007PciInit();

…

tdrv007PciInit();

…
```

4 I/O Functions

4.1 open()

NAME

open() - open a device or file.

SYNOPSIS

```
int open
(
        const char   *name,
        int          flags,
        int          mode
)
```

DESCRIPTION

Before I/O can be performed to the TDRV007 device, a file descriptor must be opened by invoking the basic I/O function *open()*.

PARAMETER

name

Specifies the device which shall be opened, the name specified in tdrv007DevCreate() must be used

flags

Not used

mode

Not used

EXAMPLE

```
int  fd;

…

/*----------------------------------------
  Open the device named "/tdrv007/0" for I/O
  ----------------------------------------*/
fd = open("/tdrv007/0", 0, 0);

…
```

RETURNS

A device descriptor number, or ERROR if the function fails an error code will be stored in *errno*.

ERROR CODES

The error codes are stored in *errno* and can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual).

SEE ALSO

ioLib, basic I/O routine - *open()*

4.2 close()

NAME

close() – close a device or file

SYNOPSIS

STATUS close
(
    int   fd
)

DESCRIPTION

This function closes opened devices.

PARAMETER

fd

This file descriptor specifies the device to be closed. The file descriptor has been returned by the *open()* function.

EXAMPLE

```
int  fd;
int  retval;

…

/*----------------
  close the device
  ----------------*/
retval = close(fd);

…
```

RETURNS

OK or ERROR if the function fails, an error code will be stored in *errno*.

ERROR CODES

The error codes are stored in *errno* and can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual).

SEE ALSO

ioLib, basic I/O routine - close()

4.3 ioctl()

NAME

ioctl() - performs an I/O control function.

SYNOPSIS

#include “tdrv007.h”

```
int ioctl
(
        int     fd,
        int     request,
        int     arg
)
```

DESCRIPTION

Special I/O operation that do not fit to the standard basic I/O calls (read, write) will be performed by calling the ioctl() function.

PARAMETER

fd

This file descriptor specifies the device to be used. The file descriptor has been returned by the *open()* function.

request

This argument specifies the function that shall be executed. Following functions are defined:

Function	Description
FIO_TDRV007_READ	Read an ARCNET message
FIO_TDRV007_WRITE	Send an ARCNET message
FIO_TDRV007_MAP	Build a map of online nodes
FIO_TDRV007_ONLINE	Configure node and set online
FIO_TDRV007_OFFLINE	Set node offline
FIO_TDRV007_DIAG	Read number of reconfigurations
FIO_TDRV007_FLUSH	Flush receive buffer
FIO_TDRV007_TXSTATUS	Get transmit state

arg

This parameter depends on the selected function (request). How to use this parameter is described below with the function.

RETURNS

Function dependent value (described with the function) or ERROR if the function fails an error code will be stored in *errno*.

ERROR CODES

The error codes are stored in *errno* and can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual) or a driver set code. Function specific error codes will be described below with the function.

Error code	Description
S_tdrv007Drv_ICMD	Undefined request specified

SEE ALSO

ioLib, basic I/O routine - ioctl()

4.3.1 FIO_TDRV007_READ

This I/O control function reads an ARCNET message. The function specific control parameter **arg** is a pointer to a *TDRV007_MSG* structure which contains the read data and special flags for execution of the function.

The structure (*TDRV007_MSG*) has the following layout and is defined in tdrv007.h:

```
typedef struct
{
    unsigned char     SID;
    unsigned char     DID;
    unsigned short    len;
    unsigned char     data[TDRV007_MAX_LONG_MSG];
    unsigned int      io_flags;
    int               timeout;
} TDRV007_MSG;
```

SID

This value returns the source ID of the message. It is the node ID of the node the message has been sent from.

DID

This value contains the destination ID of the message. This should be the node ID of the current device or 0 if the message has been sent as broadcast message.

len

This value returns the number of received data bytes.

data[]

This is the buffer which contains the data of the read message.

io_flags

This value is an ORed value of the following flags defined in “tdrv007.h”:

Flag	Description
TDRV007_F_FLUSH	Flush the read buffer and wait for the next message reception

timeout

This value specifies the maximum time the function will wait for message reception. The value is specified in system ticks.

EXAMPLE

```
#include "tdrv007.h"

…

int                fd;
TDRV007_MSG        msgBuf;
int                retval;

…

/*-----------------------------------------
  Read a message, wait a max. time of 300 ticks
  -----------------------------------------*/
msgBuf.io_flags = 0;
msgBuf.timeout = 300;

retval = ioctl(fd, FIO_TDRV007_READ, (int)&msgBuf);
if (retval != ERROR)
{
     /* function succeeded */
     printf("    Packet from node %d with bytes %d Bytes:\n",
          msgBuf.SID,
          msgBuf.len);
     printf("    Text:");
     for (i = 0; i < msgBuf.len; i++)
     {
          printf("%c", msgBuf.data[i]);
     }
}
else
{
     /* handle the error */
}

…
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

S_tdrv007Drv_NREADY	Device is not online
S_tdrv007Drv_TIMEOUT	Read access has exceeded the specified timeout
S_tdrv007Drv_IO	A hardware detected error has occurred

4.3.2 FIO_TDRV007_WRITE

This I/O control function sends an ARCNET message. The function specific control parameter **arg** is a pointer to a *TDRV007_MSG* structure which contains the message information to send and special flags for execution of the function.

The structure (*TDRV007_MSG*) has the following layout and is defined in tdrv007.h:

```
typedef struct
{
        unsigned char     SID;
        unsigned char     DID;
        unsigned short    len;
        unsigned char     data[TDRV007_MAX_LONG_MSG];
        unsigned int      io_flags;
        int               timeout;
} TDRV007_MSG;
```

SID

This value is not used.

DID

This value specifies the node number the message is addressed to or '0' if a broadcast message shall be send.

len

This value specifies the number of data bytes stored in data[]. The maximum length for short messages is 253 Byte and for long messages is 508 Byte.

data[]

This is the buffer which contains the data to be sent.

io_flags

This value is an ORed value of the following flags defined in "tdrv007.h":

Flag	Description
TDRV007_F_NOWAIT	If this flag is set the write function will not wait for message acknowledge.

timeout

This value specifies the maximum time the function will wait to send the message. The value is specified in system ticks.

EXAMPLE

```
#include "tdrv007.h"

…

int                fd;
TDRV007_MSG        msgBuf;
int                retval;

…

/*-------------------------------------------------------
  Send a message to node 100, wait a max. time of 300 ticks
  -------------------------------------------------------*/
msgBuf.DID = 100;
msgBuf.len = 5;
msgBuf.data[0] = ‘A’;
msgBuf.data[1] = ‘B’;
msgBuf.data[2] = ‘C’;
msgBuf.data[3] = ‘D’;
msgBuf.data[4] = ‘E’;
msgBuf.io_flags = 0;
msgBuf.timeout = 300;

retval = ioctl(fd, FIO_TDRV007_WRITE, (int)&msgBuf);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}

…
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

S_tdrv007Drv_NREADY	Device is not online
S_tdrv007Drv_TIMEOUT	Write access has exceeded the specified timeout
S_tdrv007Drv_NOTACK	The message has not been acknowledged

4.3.3 FIO_TDRV007_MAP

This I/O control function returns a network map. The function specific control parameter **arg** is a pointer to an array of 32 bytes where the map information will be stored.

Every bit in the array represents a possible node in the net. A set bit represents an online node, a reset bit shows that the node is offline or not present. The assignment of nodes to the bits inside the array is made in the following way:

Bit / Index	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	8	9	10	11	12	13	14	15
2	16	17	18	19	20	21	22	23
…	…	…	…	…	…	…	…	…
30	240	241	242	243	244	245	246	247
31	248	249	250	251	252	253	254	255

EXAMPLE

```
#include "tdrv007.h"

int                fd;
unsigned char      map[32];
int                retval;
int                i;

/*---------------
  Get network map
  ---------------*/
retval = ioctl(fd, FIO_TDRV007_MAP, (int)&map[0]);
if (retval != ERROR)
{
     /* function succeeded */
     for (i = 0; i < 32; i++)
     {
          printf("%02X ", map[i]);
     }
}
else
{
     /* handle the error */
}

…
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

S_tdrv007Drv_NREADY	Device is not online
S_tdrv007Drv_NETWORK_DOWN	Write access has exceeded the specified timeout

4.3.4 FIO_TDRV007_ONLINE

This I/O control function configures the ARCNET node and sets the node online. The function specific control parameter **arg** is a pointer to a *TDRV007_CONFIG* structure which contains the information how to configure the node.

The structure (*TDRV007_CONFIG*) has the following layout and is defined in tdrv007.h:

```
typedef struct
{
        unsigned int        io_flags;
        unsigned char       node_id;
        unsigned char       network_timeout;
        unsigned char       speed;
} TDRV007_CONFIG;
```

io_flags

This value is an ORed value of the flags described below and defined in “tdrv007.h”:

Value	Description
TDRV007_F_BROADCAST	If set, the controller will be configured to accept broadcast messages from the network.
TDRV007_F_LONGPACKET	If set the controller will be configured to receive both short and long packets. If not set only short packets can be received.
TDRV007_F_NODE_ID	If set the node ID specified in TDRV007_CONFIG will be used for the node. If not set, the node ID chosen by the hardware DIP switch will be used.

node_id

This value selects the node ID that will be configured if *TDRV007_F_NODE_ID* is configured in *io_flag*.

network_timeout

This value specifies the network timeout configuration to be set up. Allowed configuration values are shown in the table below:

Value	Response Time	Idle Time	Reconfig Time
0	596.8 µs	656 µs	840 ms
1	298.4 µs	328 µs	840 ms
2	74.7 µs	82 µs	840 ms
3	37.35 µs	41 µs	420 ms

speed

This value specifies the network speed. The following values are valid and predefined in "tdrv007.h":

Speed	Description
TDRV007_312KBTS	312.5 kbps
TDRV007_625KBTS	625.0 kbps
TDRV007_1250KBTS	1.25 Mbps
TDRV007_2500KBTS	2.5 Mbps
TDRV007_5000KBTS	5 Mbps

EXAMPLE

```
#include "tdrv007.h"

int                fd;
TDRV007_MSG        cfgBuf;
int                retval;

…

/*-------------------------------------------------
  Set node online: node Id: 33
        accept broadcast and accept long messages
        2.5 Mbps and network timeout configuration 3
  -------------------------------------------------*/
cfgBuf.io_flags = TDRV007_F_BROADCAST |
                  TDRV007_F_LONGPACKET |
                  TDRV007_F_NODE_ID;
cfgBuf.node_id = 33;
cfgBuf.network_timeout = 3;
cfgBuf.speed = TDRV007_2500KBTS;

retval = ioctl(fd, FIO_TDRV007_ONLINE, (int)&cfgBuf);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}

…
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

S_tdrv007Drv_IO	Device initialization failed, hardware returned an unexpected value
S_tdrv007Drv_ISPEED	The specified speed value is invalid
S_tdrv007Drv_ITYPE	The detected module type is not supported, can't decide to use RS485 or hybrid setup.
S_tdrv007Drv_DUPID	The node ID specified or read from the DIP-switch is already online on another node in the network
S_tdrv007Drv_NETWORK_DOWN	The node can't get online

4.3.5 FIO_TDRV007_OFFLINE

This I/O control function removed the ARCNET node from the network and sets it to offline mode. There is no function specific control parameter **arg** used.

EXAMPLE

```
#include "tdrv007.h"

…

int           fd;
int           retval;

…

/*--------------------------
  Remove device from network
  --------------------------*/
retval = ioctl(fd, FIO_TDRV007_OFFLINE, 0);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}

…
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

No special error codes defined.

4.3.6 FIO_TDRV007_DIAG

This I/O control function returns diagnostic information. The function specific control parameter **arg** is a pointer to an int value where the number of reconfigurations will be stored. After executing the function the reconfiguration counter will be reset.

EXAMPLE

```
#include "tdrv007.h"

…

int           fd;
int           recons;
STATUS        retval;

/*--------------------------
  Get diagnostic information
  --------------------------*/
retval = ioctl(fd, FIO_TDRV007_DIAG, (int)&recons);
if (retval != ERROR)
{
     /* function succeeded */
     printf("Number of reconfigurations: %d\n", recons);
}
else
{
     /* handle the error */
}

…
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

No special error codes defined.

4.3.7 FIO_TDRV007_FLUSH

This I/O control function flushes the receive buffer. There is no function specific control parameter **arg** used.

EXAMPLE

```
#include "tdrv007.h"

…

int           fd;
int           retval;

…

/*------------------------
  Flush the receive buffer
  ------------------------*/
retval = ioctl(fd, FIO_TDRV007_FLUSH, 0);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}

…
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

No special error codes defined.

4.3.8 FIO_TDRV007_TXSTATUS

This I/O control function returns the current transmit state of the device. The function specific control parameter **arg** is a pointer to an int value where the transmit status will be stored.

The transmit state may return the following values, defined in 'tdrv007.h':

S_tdrv007Drv_BUSY	The message transmission is active
S_tdrv007Drv_NOTACK	The message is not acknowledged
S_tdrv007Drv_TIMEOUT	The message transmission times out
OK	OK

EXAMPLE

```
#include "tdrv007.h"

…

int           fd;
int           txState;
int           retval;

/*--------------------------
  Get transmit status
  --------------------------*/
retval = ioctl(fd, FIO_TDRV007_TXSTATUS, (int)&txState);
if (retval != ERROR)
{
     /* function succeeded */
     printf("Write State: %d\n", txState);
}
else
{
     /* handle the error */
}

…
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

No special error codes defined.