The Embedded I/O Company

TEWS TECHNOLOGIES

TIP815-SW-42

VxWorks Device Driver

ARCNET Interface IP

Version 1.2

User Manual

Issue 1.2

November 2003

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

TIP815-SW-42

ARCNET Interface IP

VxWorks Device Driver

Issue	Description	Date
1.0	First Issue	June 1995
1.1	New chapter “Installation”	May 1998
1.2	General Revision	November 2003

Table of Content

1 Introduction

The TIP815-SW-42 VxWorks device driver software allows the operation of the TIP815 IP conforming to the VxWorks I/O system specification. This includes a device-independent interface with *open*(), *read(), write()* and *ioctl()* functions.

After installation of the device driver software to the I/O system, messages can be transmitted to and received from the ARCNET interface by calling the *write()* or *read()* 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 queuing mechanism to prevent simultaneous requests by multiple users from interfering with each other.

2 Installation

The software is delivered on a 3½" HD diskette.

Following files are located on the diskette:

t815drv.c	TIP815 Driver Source
tip815.h	TIP815 Driver Include File for driver and application
t815defs.h	TIP815 Internal Driver Include File
arcnet.h	ARCNET controller programming model
ipchip.h	IPIC programming model definitions
t815exam.c	TIP815 Driver usage example program

For installation the files have to be copied to the desired target directory.

2.1 TIP815 mounted on MVME162

The device driver support the on board IP slots of the Motorola MVME162 by default. That means the IP chip will be set by the driver if one of the specific I/O addresses is selected when creating the drive.

2.2 TIP815 mounted to another Carrier

The device driver software supports also TEWS IP carrier boards and others. The system has to be setup to guarantee the following points:

- The modules I/O area must be accessible (address decoder / MMU).
- The interrupts of the TIP815 must be connected to the CPU and must be handled by the hardware.

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 t815Drv()

NAME

t815Drv() - installs the TIP815 driver in the I/O system.

SYNOPSIS

void t815Drv(void)

DESCRIPTION

This function installs the TIP815 driver in the I/O system.

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.

RETURNS

N/A

3.2 t815DevCreate()

NAME

t815DevCreate() - adds a TIP815 device to the system and initialize device hardware.

SYNOPSIS

```
STATUS t815DevCreate
    (
    char        *name,      /* name of the device to create     */
    uint        addr,       /* physical device address          */
    uint        vector,     /* interrupt vector                 */
    uint        level       /* interrupt level [1..7]           */
    )
```

DESCRIPTION

This routine is called to add a device to the system that will be serviced by the TIP815 driver. This function must be called before performing any I/O request to this device. After device creation the ARCNET controller is **OFFLINE** that means no messages can be received or transmitted. To set the ARCNET controller **ONLINE** the special I/O control command *FIOONLINE* must be executed.

There are several device dependent arguments required for device initialization and allocation of system resources.

PARAMETER

name	Name of the device to create (i.e.”/t815A”)
addr	Address of TIP815 (any valid address of an internal IP slot (A..D) or an external carrier board on the VMEbus)
vector	The vector number on which the ARCNET controller will generate interrupts
level	Defines the interrupt level

EXAMPLE

```
#include “tip815.h”

...

/*-----------------------------------------------------------------------
   Create a device "/t815A" for TIP815 IP on slot A
   Generate interrupts on interrupt level 4 with vector 0x88
  -----------------------------------------------------------------------*/
status = t815DevCreate ("/t815A",
                        0xFFF58000,
                        0x88,
                        4);
...
```

RETURNS

OK or ERROR

INCLUDE FILES

tip815.h

4 I/O interface functions

This chapter describes the interface to the basic I/O system used for communication over the ARCNET interface.

4.1 open()

NAME

open() - opens a device or file

SYNOPSIS

```
int open
        (
        const char  *name,          /* name of the device to open              */
        int         flags,          /* not used for TIP815 driver, must be 0   */
        int         mode            /* not used for TIP815 driver, must be 0   */
        )
```

DESCRIPTION

Before I/O can be performed to the TIP815 device, a file descriptor must be opened by invoking the basic I/O function *open().*

EXAMPLE

```
...

/*------------------------------------------------------------
  Open the device named "/t815A" for I/O
  ------------------------------------------------------------*/
fd = open <Tab>("/t815A", 0, 0);

...
```

RETURNS

A device descriptor number or ERROR (if the device does not exist or no device descriptors are available)

SEE ALSO

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

4.2 read()

NAME

read() – reads a message from specified TIP815 device

SYNOPSIS

```
int read
        (
        int         fd,         /* device descriptor from opened TIP815 device   */
        char        *buffer,    /* pointer to message buffer                     */
        size_t      maxbytes    /* not used                                      */
        )
```

DESCRIPTION

Once a TIP815 device has been opened, tasks can read messages from the device.

PARAMETER

The argument **fd** specifies the LON device.

The argument **buffer** points to a data structure holding the message information. The data structure contains a timeout value, special I/O flags and the received data packet with the source and the destination ID, the message length, and received data bytes (see below).

The argument **maxbytes** is unused.

Data structure *T815_MSG:*

```
typedef struct
{
    unsigned char   SID;                    /* Source ID          */
    unsigned char   DID;                    /* Destination ID or 0 for broadcasts          */
    unsigned short  len;                    /* Message data length      */
    unsigned char   data[MAX_LONG_MSG];     /* Message data:  253 bytes for short packets  */
                                            /* 508 bytes for long packets          */
    unsigned int    io_flags;               /* I/O flags [ T815_FLUSH] */
    int             timeout;                /* NO_WAIT, WAIT_FOREVER or number of ticks  */
} T815_MSG;
```

For a read request to the device it is only necessary to specify desired I/O flags in **io_flags** and the amount of time in ticks the task shall wait for a message in **timeout.** A timeout value of *WAIT_FOREVER* means wait indefinitely; a value of *NO_WAIT* means do not wait at all. If the device is blocked by another read request or no message is available in the read buffer, the requesting task will be blocked until a message was read or the request times out.

The TIP815 device driver uses a global mechanism for returning status codes when an error occurs. A global integer variable *errno* is set to an appropriate error code. The global variable *errno* is never cleared by the TIP815 device driver. Thus, its value always indicates the last error status set.

After successful completion of a read request **SID** contains the source ID, **DID** contains the destination ID, **len** the number of received data bytes, and **data[]** contains the received message.

EXAMPLE

```
#include <vxworks.h>
#include <errnoLib.h>
#include "tip815.h"

...

T815_MSG      rcvbuf;
int           nbytes;
int           fd;

...

fd = open ("/t815A", 0, 0);

/*-------------------------------------------------------
  Read a message from opened TIP815 device.
    o flush the input ring buffer before reading
    o if there is no message in the read buffer, the read
      request times out after 500 ticks
  -------------------------------------------------------*/
rcvbuf.io_flags    = T815_FLUSH;
rcvbuf.timeout     = 500;                       /* ticks */
nbytes             = read (fd, (char *)&rcvbuf, 0);
if (nbytes == ERROR)
{
  /* check errno for returned error codes */
}
else
{
  /* process message */
}

...
```

RETURNS

ERROR or number of *data bytes* read

INCLUDE FILES

vxworks.h

tip815.h

errnoLib.h

SEE ALSO

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

4.3 write()

NAME

write() – writes a message to specified TIP815 device

SYNOPSIS

```
int write
        (
        int             fd          /* device descriptor from opened TIP815 device                  */
        char            *buffer,    /* pointer to message buffer                                    */
        size_t          nbytes      /* not used because this information is part of the T815_MSG     */
                                    /* structure                                                    */
        )
```

DESCRIPTION

This routine writes a message to the specified TIP815 device.

PARAMETER

The argument **fd** specifies the device descriptor to select the TIP815 device.

The argument **buffer** points to a data structure holding the message information. The data structure contains a timeout value, special I/O flags and the data packet with the source and the destination ID, the message length, and the message data (see below).

The argument **nbytes** is unused.

Data structure *T815_MSG:*

```
typedef struct
{
    unsigned char   SID;                    /* Source ID                                    */
    unsigned char   DID;                    /* Destination ID or 0 for broadcasts           */
    unsigned short  len;                    /* Message data length      */
    unsigned char   data[MAX_LONG_MSG];     /* Message data:  253 bytes for short packets   */
                                            /* 508 bytes for long packets                   */
    unsigned int    io_flags;               /* I/O flags [ T815_NOWAIT]                     */
    int             timeout;                /* NO_WAIT, WAIT_FOREVER or number of ticks     */
} T815_MSG;
```

The message parameter **timeout** specifies the amount of time in ticks that the task will wait for completion of the write request. A timeout value of *WAIT_FOREVER* means wait indefinitely; a value of *NO_WAIT* means do not wait at all.

The only valid I/O flag set in **io_flags** is *T815_NOWAIT*. If this flag is set, *write()* returns immediately after starting transmission, otherwise *write()* waits until the message is transmitted and acknowledged or an error occurs.

The parameter **len** specifies the number of data bytes in **data[]** to be transferred. The number of data bytes is limited to 253 bytes for short packets and 508 bytes for long packets.

The parameter **DID** specifies the destination ID (destination node hardware address).

The parameter **SID** is not used.

If the device is blocked by another write request, the requesting task will be blocked until the transmitter becomes free or the request times out.

The TIP815 device driver uses a global mechanism for returning status codes when an error occurs. A global integer variable *errno* is set to an appropriate error code. The global variable *errno* is never cleared by the TIP815 device driver. Thus, its value always indicates the last error status set.

EXAMPLE

```
#include "tip815.h"
#include "errnoLib.h"
#include "vxworks.h"

#define   HELLO     "HELLOOOO WORLD"

...

T815_MSG  sndbuf;
int       nbytes;
int       fd;

...

fd = open ("/t815A", 0, 0);

/*------------------------------------------------------------
  Write a message to a TIP815 device.
  o if there is a problem with the transmitter the write
    request times out after 100 ticks
  o return immediately after starting transmission (T815_NOWAIT)
  ------------------------------------------------------------*/
sndbuf.io_flags    = T815_NOWAIT;
sndbuf.timeout     = 100;    /* ticks */
sndbuf.DID         = 123;
sndbuf.len         = 14;
memcpy(sndbuf.data, HELLO, 14);
nbytes = write(fd, (char *)&sndbuf, 0);

if (nbytes == ERROR)
{
/* check errno for returned error codes */
}

...
```

RETURNS

ERROR or number of *data bytes* written

INCLUDE FILES

tip815.h
errnoLib.h
vxworks.h

SEE ALSO

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

4.4 ioctl()

NAME

ioctl() - performs an I/O control function

SYNOPSIS

```
int ioctl
(
int     fd,             /* device descriptor from opened TIP815 device*/
int     function,       /* function code                              */
int     arg             /* optional function dependent argument       */
)
```

DESCRIPTION

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

PARAMETER

The parameter **fd** specifies the device descriptor to select the TIP815 device.

The parameter **function** selects the action which will be executed by the driver and an optional function dependent argument in **arg** (see description below).

4.4.1 FIODIAG Get the number of node reconfigurations

The number of node reconfigurations reflects the quality of the network. If the number grows up soon there is something wrong on the network. The number of reconfiguration is copied to an integer variable, referenced by a pointer passed to the driver in the function dependent argument **arg**. The reconfiguration counter will be reset after reading.

4.4.2 FIOMAP Build a map of the network

A pointer to a buffer is passed to the driver in the function dependent argument **arg**. Bits in the buffer are set for present or reset for absent nodes. The buffer must be 32 bytes (256 bits) in length. The first byte in the array presents nodes 0 through 7; the second byte presents 8 through 15, and so on. The least significant bit of the byte presents the smallest node number.

4.4.3 FIOFLUS Flush the read buffer

This I/O control function flushes the read ring buffer. No arguments are required for this call. The function dependent argument **arg** can be set to zero.

4.4.4 FIOOFFLINE Shut down the node and remove it from network

This function shuts down the node and removes it from the network. This function needs no argument. The parameter **arg** can be set to zero.

4.4.5 FIOONLINE Initialize controller and enter network

After device creation (*t815DevCreate()*) the ARCNET controller is in reset state and not connected to the network. The I/O control function *FIOONLINE* sets up the ARCNET controller using information specified in the *T815_CONFIG* structure (see description below). A pointer to the *T815_CONFIG* structure is passed to the driver by the function dependent argument **arg**.

Data structure *T815_CONFIG:*

```
typedef struct
{
    unsigned int        io_flags;       /* I/O flags: [T815_BROADCAST |          */
                                        /* T815_LONGPACKET | T815_NODE_ID]       */
    unsigned char       node_id;        /* node ID          */
    unsigned char       network_timeout;
    unsigned char       speed;          /* network speed */
} T815_CONFIG;
```

Possible I/O flags specified in **io_flags** are:

T815_BROADCAST	Configure the controller to accept broadcast messages from the network.
T815_LONGPACKET	Configure the controller to receive both short and long packets. If not set only short packets can be received.
T815_NODE_ID	Use node ID specified in *T815_CONFIG*. If not set, determine the node ID by reading the hardware DIP switch.

The parameter **node_id** specifies the node address on the network for this node. This node ID is only used if the I/O flag *T815_NODE_ID* is set. Valid values are 0 to 255.

The parameter **network_timeout** specifies the Response, Idle and Recon Times of the ARCNET controller. All nodes should be configured with the same timeout value for proper network operation. Valid values are shown in the table:

For TIP815-10 / -20:

Value	Response Time	Idle Time	Reconfig Time
0	1130 µs	1237 µs	1680 ms
1	563 µs	624 µs	1680 ms
2	285 µs	316 µs	1680 ms
3	78 µs	86 µs	840 ms

For TIP815-11 / -21:

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

The last parameter **speed** determines the network speed by setting the clock prescaler to one of five possible values:

Name	Value	Network Speed	Module Types
T815_312KBTS	0x03	5 Mbps	-10 / -11 / -20 / -21
T815_625KBTS	0x02	2.5 Mbps	-10 / -11 / -20 / -21
T815_1250KBTS	0x01	1.25 Mbps	-10 / -11 / -20 / -21
T815_2500KBTS	0x00	625 Kbps	-10 / -11 / -20 / -21
T815_5000KBTS	0xFF	312.5 Kbps	-11 / -21

After successful execution of *FIOONLINE* the ARCNET controller is connected to the network and messages can be received and transmitted over the network.

4.4.6 FIOTXSTATUS Get actual transmission status

This I/O control function is useful for write request with I/O flag *T815_NOWAIT* set to query the result of the previous transmission. The transmission status is returned in the global variable *errno*.

Possible returned status (error) codes are:

OK	transmission successful completed
S_t815Dev_BUSY	transmission in progress
S_t815Dev_NOTACK	no ACK from receiver after transmission
S_t815Dev_TIMEOUT	transmission times out

4.4.7 EXAMPLES for Control Functions

```
#include "errnoLib.h"
#include "iolib.h"
#include "tip815.h"

int           status;
int           recon;
unsigned char map[32];
int           fd;
T815_CONF     conf;

...

fd = open <Tab>("/t815A", 0, 0);

/*-------------------------------------------------------
  Initialize the ARCNET controller and entering the network
  o accept broadcast messages from the network
  o receive both short and long packets
  o use node ID specified in node_id
  o setup network speed to 2.5 Mbps
  o response time = 1130 µs
  o idle time = 1237 µs
  o reconfig time = 1680 ms
  -------------------------------------------------------*/
conf.io_flags =        T815_BROADCAST |
                       T815_LONGPACKET |
                       T815_NODE_ID;
conf.node_id           = 9;
conf.network_timeout   = 0;
conf.speed             = 0;
status = ioctl(fd, FIOONLINE, (int)&conf);

...
/*----------------------------------
  Get number of node reconfigurations
  ----------------------------------*/
status = ioctl(fd, FIODIAG, (int)&recon);
printf(" %d reconfigurations occurred since last FIODIAG call\n", recon);

...
```

```
/*-------------------------------------------
  Build a map of nodes contained on the network
  -------------------------------------------*/
status = ioctl(fd, FIOMAP, (int)map);

if (map[1] & 0x02) printf(" node 9 is available \n");

...

/*-------------------------
  Flush the read ring buffer
  -------------------------*/
status = ioctl(fd, FIOFLUSH, 0);

...

/*-----------------------------
  Get actual transmission status
  -----------------------------*/
status = ioctl(fd, FIOTXSTATUS, 0);
if (errnoGet() == OK) printf(" message successful sent\n");
if (errnoGet() == S_t815Dev_BUSY) printf(" transmission in progress\n");

...

/*-----------------------------------------------------
  Shut down the node hardware and remove the node from the
  network
  -----------------------------------------------------*/
status = ioctl(fd, FIOOFFLINE, 0);

...
```

RETURNS

OK or ERROR (if the device descriptor does not exist or the function code is unknown)

INCLUDE FILES

tip815.h

errnoLib.h

iolib.h

SEE ALSO

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

5 Appendix

5.1 Predefined Symbols

This chapter describes the symbols which are defined in the file *tip815.h*.

5.1.1 ioctl function codes

FIOMAP	100	Build a map of nodes contained on the network
FIOONLINE	101	Initialize the ARCNET controller and entering the network
FIOFFLINE	102	Shut down the node hardware and remove the node from the network
FIODIAG	103	Get number of node reconfigurations
FIOTXSTATUS	104	Get actual transmitter status

5.1.2 I/O flags

T815_FLUSH	(1 << 0)	Flush read ring buffer
T815_BROADCAST	(1 << 1)	Configure the controller to accept broadcast messages from the network
T815_LONGPACKET	(1 << 2)	Configure the controller to receive both short and long packets. If not set only short packets can be received.
T815_NODE_ID	(1 << 3)	Use node ID specified in *T815_CONFIG*. If not set, determine the node ID by reading the hardware DIP switch.
T815_NOWAIT	(1 << 4)	Return immediately after starting transmission selectable bus speeds

5.1.3 Packet sizes

MAX_SHORT_MSG	253	Length of short packets in bytes
MAX_LONG_MSG	508	Length of long packets in bytes

5.1.4 Network speeds

T815_312Kbts	0x03	Transfer rate is set to 312.5 Kbts/s
T815_625Kbts	0x02	Transfer rate is set to 650 Kbts/s
T815_1250Kbts	0x01	Transfer rate is set to 1.25 Mbts/s
T815_2500Kbts	0x00	Transfer rate is set to 2.5 Mbts/s
T815_5000Kbts	0xFF	Transfer rate is set to 5 Mbts/s (only TIP815-11/TIP815-21)

5.2 Status and Error Codes

If the device driver creates an error the error codes are stored in the *errno*. They can be read with the VxWorks function *errnoGet()* or *printErrno().*

S_t815Drv_NXIO	0x08150001	No TIP815 device found at specified address
S_t815Dev_ICMD	0x08150002	Illegal *ioctl* command
S_t815Dev_BUSY	0x08150003	Transmitter is busy
S_t815Dev_NOTACK	0x08150004	No ACK from receiver after transmission
S_t815Dev_ TIMEOUT	0x08150005	I/O request times out
S_t815Dev_QFULL	0x08150006	Queue for incoming messages is full
S_t815Dev_BAD_PACKET_SIZE	0x08150007	Illegal number of bytes to transfer
S_t815Dev_DUPID	0x08150008	Illegal (duplicate) node ID
S_t815Dev_NETWORK_DOWN	0x08150009	No activity on network
S_t815Dev_IO	0x0815000A	Device I/O error
S_t815Dev_NREADY	0x0815000B	Device is not initialized (FIOONLINE ?)
S_t815Dev_ISPEED	0x0815000C	Illegal bit rate selected