The Embedded I/O Company

TEWS TECHNOLOGIES

TIP815-SW-72

LynxOS Device Driver

ARCNET Controller

Version 2.0.x

User Manual

Issue 2.0.0

November 2004

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-72

ARCNET Controller

LynxOS Device Driver

Issue	Description	Date
1.0	First Issue	March 1, 2003
1.1	Correction of definition description	November 28, 2003
2.0.0	Adaptation to CARRIER-SW-72	November 18, 2004

Table of Content

1 Introduction

The TIP815-SW-72 LynxOS device driver allows the operation of a TIP815 ARCNET controller IP on LynxOS operating systems.

Because the TIP815 device driver is stacked on the TEWS TECHNOLOGIES IPAC Carrier Driver, it is necessary to install also the IPAC Carrier Driver. Please refer to the IPAC Carrier Driver user manual for further information.

The standard file (I/O) functions (open, close, read, write, ioctl) provide the basic interface for opening and closing a file descriptor and for performing device I/O and configuration operations.

The TIP815 device driver includes the following functions:

- Receiving and Sending messages
- Setup ARCNET controller and getting online
- Getting Offline
- Flushing receive buffer
- Getting diagnostic value

To understand all features of this device driver, it is recommended to read the TIP815 User Manual.

2 Installation

The software is delivered on a PC formatted 3½" HD diskette.

The directory A:\TIP815-SW-72 contains the following files:

TIP815-SW-72.pdf	This manual in PDF format
TIP815-SW-72.tar	Device Driver and Example sources

The TAR archive TIP815-SW-72.tar contains the following files and directories:

tip815/tip815.c	Driver source code
tip815/tip815.h	Definitions and data structures for driver and application
tip815/tip815def.h	Definitions and data structures for the driver
tip815/arcdef.h	Definitions of controller values
tip815/tip815_info.c	Device information definition
tip815/tip815_info.h	Device information definition header
tip815/tip815.cfg	Driver configuration file include
tip815/tip815.import	Linker import file
tip815/Makefile	Device driver make file
tip815/Example/example.c	Example application source
tip815/Example/Makefile	Example application make file

In order to perform a driver installation first extract the TAR file to a temporary directory then copy the following files to their target directories:

1. Create a new directory in the system drivers directory path /sys/drivers.xxx, where xxx represents the BSP that supports the target hardware.

   For example: /sys/drivers.pp_drm/tip815 or /sys/drivers.cpci_x86/tip815

2. Copy the following files to this directory:
   - tip815.c
   - tip815def.h
   - tip815.import
   - arcnet.h
   - Makefile

3. Copy tip815.h to /usr/include/

4. Copy tip815_info.c to /sys/devices.xxx/ or /sys/devices if /sys/devices.xxx does not exist (xxx represents the BSP).

5. Copy tip815_info.h to /sys/dheaders/

6. Copy tip815.cfg to */sys/cfg.xxx/*, where xxx represents the BSP for the target platform

For example: /sys/cfg.ppc or /sys/cfg.x86 ....

Before building a new device driver, the TEWS TECHNOLOGIES IPAC Carrier Driver must be installed properly, because this driver includes the header file *ipac_carrier.h*, which is part of the IPAC Carrier Driver distribution. Please refer to the IPAC carrier driver user manual in the directory path *A:\CARRIER-SW-72* on the separate distribution diskette.

2.1 Device Driver Installation

The two methods of driver installation are as follows:

- Static Installation
- Dynamic Installation (only native LynxOS systems)

Both installation methods require the TEWS TECHNOLOGIES IPAC Carrier Driver. Please refer to the IPAC Carrier Driver User Manual for detailed information.

2.1.1 Static Installation

With this method, the driver object code is linked with the kernel routines and is installed during system start-up.

2.1.1.1 Build the driver object

1. Change to the directory /sys/drivers.xxx/tip815, where xxx represents the BSP that supports the target hardware.
2. To update the library /sys/lib/libdrivers.a enter:

   ```
   make install
   ```

2.1.1.2 Create Device Information Declaration

1. Change to the directory /sys/devices.xxx/ or /sys/devices if /sys/devices.xxx does not exist (xxx represents the BSP).
2. Add the following dependencies to the Makefile

   ```
   DEVICE_FILES_all = ... tip815_info.x
   ```

   And at the end of the Makefile

   ```
   tip815_info.o:$(DHEADERS)/tip815_info.h
   ```

3. To update the library /sys/lib/libdevices.a enter:

   ```
   make install
   ```

2.1.1.3 Modify the Device and Driver Configuration File

In order to insert the driver object code into the kernel image, an appropriate entry in file CONFIG.TBL must be created.

1. Change to the directory /sys/lynx.os/ respective /sys/bsp.xxx, where xxx represents the BSP that supports the target hardware.
2. Create an entry at the end of the file CONFIG.TBL

   Insert the following entry at the end of this file. Be sure that the necessary TEWS TECHNOLOGIES IPAC Carrier Driver is included **before** this entry.

   ```
   I:tip815.cfg
   ```

2.1.1.4 Rebuild the Kernel

1. Change to the directory /sys/lynx.os/ (/sys/bsp.xxx)
2. Enter the following command to rebuild the kernel:

   ```
   make install
   ```

3. Reboot the newly created operating system by the following command (not necessary for KDIs):

   ```
   reboot –aN
   ```

   The N flag instructs init to run mknod and create all the nodes mentioned in the new nodetab.

4. After reboot you should find the following new devices (depends on the device configuration): /dev/tip815_0, /dev/tip815_1, /dev/tip815_2, …

2.1.2 Dynamic Installation

This method allows you to install the driver after the operating system is booted. The driver object code is attached to the end of the kernel image and the operating system dynamically adds this driver to its internal structures. The driver can also be removed dynamically.

2.1.2.1 Build the driver object

1. Change to the directory /sys/drivers.xxx/tip815, where xxx represents the BSP that supports the target hardware.
2. To make the dynamic link-able driver enter :

   ```
   make
   ```

2.1.2.2 Create Device Information Declaration

1. Change to the directory /sys/drivers.xxx/tip815, where xxx represents the BSP that supports the target hardware.
2. To create a device definition file for the major device (this works only on native system)

   ```
   make t815info
   ```

3. To install the driver enter:

   ```
   drinstall –c tip815.obj
   ```

   If successful, drinstall returns a unique <driver-ID>
4. To install the major device enter:

   ```
   devinstall –c –d <driver-ID> t815info
   ```

   The <driver-ID> is returned by the drinstall command
5. To create nodes for the devices enter:

   ```
   mknod /dev/tip815_0 c <major_no> 0
   mknod /dev/tip815_1 c <major_no> 1
   mknod /dev/tip815_2 c <major_no> 2
   ...
   ```

   The <major_no> is returned by the devinstall command.

If all steps are successfully completed the TIP815 is ready to use.

2.1.2.3 Uninstall dynamic loaded driver

To uninstall the TIP815 device enter the following commands:

```
devinstall –u –c <device-ID>
drinstall –u <driver-ID>
```

2.1.3 Configuration File: CONFIG.TBL

The device and driver configuration file CONFIG.TBL contains entries for device drivers and its major and minor device declarations. Each time the system is rebuild, the config utility read this file and produces a new set of driver and device configuration tables and a corresponding nodetab.

To install the TIP815 driver and devices into the LynxOS system, the configuration include file tip815.cfg must be included in the CONFIG.TBL (see also 2.1.1.3).

The file tip815.cfg on the distribution disk contains the driver entry (*C:tip815:*\....) and a major device entry (*D:TIP815:t815info::*) with 9 minor device entries (*"N: tip815_0:0", ..., "N: tip815_8:8"*).

If the driver should support more than 9 minor devices because more than 9 TIP815 are plugged, additional minor device entries must be added. To create the device node */dev/tip815_9* the line *N:tip815_9:9* must be added at the end of the file tip815.cfg. For the next node a minor device entry with 10 must be added and so on.

This example shows a driver entry with one major device and nine minor devices:

```
#    Format:
#    C:driver-name:open:close:read:write:select:control:install:uninstall
#    D:device-name:info-block-name:raw-partner-name
#    N:node-name:minor-dev

C:tip815:\
     :t815open:t815close:t815read:t815write:\
     ::t815ioctl:t815install:t815uninstall
D:TIP815:t815info::
N:tip815_0:0
N:tip815_1:1
N:tip815_2:2
N:tip815_3:3
N:tip815_4:4
N:tip815_5:5
N:tip815_6:6
N:tip815_7:7
N:tip815_8:8
```

The configuration above creates the following node in the /dev directory.

/dev/tip815_0 ... /dev/tip815_8

3 TIP815 Device Driver Programming

LynxOS system calls are all available directly to any C program. They are implemented as ordinary function calls to "glue" routines in the system library, which trap to the OS code.

Note that many system calls use data structures, which should be obtained in a program from appropriate header files. Necessary header files are listed with the system call synopsis.

3.1 open()

NAME

open() - open a file

SYNOPSIS

#include <sys/file.h>
#include <sys/types.h>
#include <fcntl.h>

int open (char *path, int oflags[, mode_t mode])

DESCRIPTION

Opens a file (TIP815 device) named in ***path*** for reading and writing. The value of ***oflags*** indicates the intended use of the file. In case of a TIP815 devices ***oflags*** must be set to **O_RDWR** to open the file for both reading and writing.

The ***mode*** argument is required only when a file is created. Because a TIP815 device already exists this argument is ignored.

EXAMPLE

```
int fd

/* open the device named "/dev/tip815_0" for I/O */
fd = open ("/dev/tip815_0", O_RDWR);
```

RETURNS

open returns a file descriptor number if successful, or –1 on error.

SEE ALSO

LynxOS System Call - open()

3.2 close()

NAME

close() – close a file

SYNOPSIS

int close(int fd)

DESCRIPTION

This function closes an opened device.

EXAMPLE

```
int  result;

...

/*
**   close the device
*/
result = close(fd);

...
```

RETURNS

close returns 0 (OK) if successful, or –1 on error

SEE ALSO

LynxOS System Call - close()

3.3 read()

NAME

read() - read from a file

SYNOPSIS

#include <tip815.h>

int read (int fd, char *buff, int count)

DESCRIPTION

This function attempts to read a message from the TIP815 associated with the file descriptor *fd* into a structure (*T815_READ_BUF*) pointed by *buff*. The argument *count* specifies the length of the buffer and must be set to the length of the structure *T815_READ_BUF*.

The *T815_READ_BUF* structure has the following layout:

```
typedef struct
{
        unsigned char       SID;
        unsigned char       DID;
        unsigned short      len;            /* Message data length                  */
        unsigned char       data[T815_MAX_LONG_MSG];  /* Message data               */
                                            /* - 253 bytes data for SHORT PACKET    */
                                            /* - 508 bytes data for LONG PACKET     */
        unsigned long       io_flags;
        int                 timeout;
} T815_READ_BUF;
```

SID

Returns the source ID. The source ID is the ID of the node, which has send the message.

DID

Returns the target ID. This ID must be the ID of the actual target.

len

Returns the length of the message.

data[]

This array contains the received ARCNET message.

io_flags

This value is an ORed value of the following flags:

T815_FLUSH	Flush receive buffer and wait for the next message receive.

timeout

Specifies the maximum time (in ticks) to wait for a message receive. If the time expires, the driver will return with a timeout error.

EXAMPLE

```
int           fd;
int           result;
T815_READ_BUF readBuf;

/* Flush buffer and read message, timeout is 100 ticks */
readBuf.io_flags   = T815_FLUSH;
readBuf.timeout    = 100;
result = read(fd, (char*)& readBuf, sizeof(readBuf));
if (result <= 0)
{
     // process error;
}
```

RETURNS

When *read* succeeds, the size of the read buffer (*T815_READ_BUF)* is returned. If read fails, -1 (SYSERR) is returned.

On error, *errno* will contain a standard read error code (see also LynxOS System Call – read) or a special error code (see also below Additional Error Codes)

SEE ALSO

LynxOS System Call - read()

TIP815 example application

3.4 write()

NAME

write() – write to a file

SYNOPSIS

#include <tip815.h>

int write (int fd, char *buff, int count)

DESCRIPTION

This function attempts to write a message with the TIP815 associated with the file descriptor *fd* from a structure (T815_WRITE_BUF) pointed by buff. The argument count specifies the length of the buffer and must be set to the length of the structure T815_WRITE_BUF.

The *T815_READ_BUF* structure has the following layout:

```
typedef struct
{
        unsigned char       SID;
        unsigned char       DID;
        unsigned short      len;            /* Message data length                  */
        unsigned char       data[T815_MAX_LONG_MSG];  /* Message data               */
                                            /* - 253 bytes data for SHORT PACKET    */
                                            /* - 508 bytes data for LONG PACKET     */
        unsigned long       io_flags;
        int                 timeout;
} T815_READ_BUF;
```

SID

This parameter is unused.

DID

Specifies the target ID. The target ID selects the node the message should be sent to.

len

Specifies the length of the message

data[]

This array must contain the message that should be sent.

io_flags

This value is an ORed value of the following flags:

T815_NOWAIT	The function will return immediately and will not wait for successful transmission.

timeout

Specifies the maximum time (in ticks) to wait for a successful message send. If the time expires, the driver will return with a timeout error.

EXAMPLE

```
int                fd;
int                result;
T815_WRITE_BUF     sndBuf;

/* Send a message to node 5, timeout is 100 ticks */
sndBuf.DID         = 5;
sndBuf.timeout     = 100;
sndBuf.io_flags    = 0;
sndBuf.len         = strlen(“Hello”);
memcpy(sndBuf.data, “Hello”, sndBuf.len);

result = write(fd, (char*)&sndBuf, sizeof(sndBuf));
if (result <= 0)
{
     // process error;
}...
```

RETURNS

When *write* succeeds, the size of the write buffer (*T815_WRITE_BUF)* is returned. If write fails, -1 (SYSERR) is returned.

On error, *errno* will contain a standard read error code (see also LynxOS System Call – read) or a special error code (see also below Additional Error Codes)

SEE ALSO

LynxOS System Call - write()

TIP815 example application

3.5 ioctl()

NAME

ioctl() – I/O device control

SYNOPSIS

#include <ioctl.h>
#include <tip815.h>

int ioctl (int fd, int request, char *arg)

DESCRIPTION

ioctl provides a way of sending special commands to a device driver. The call sends the value of request and the pointer arg to the device associated with the descriptor fd.

The following ioctl codes are supported by the driver and are defined in TIP815.h:

Symbol	Meaning
T815_C_ONLINE	Configure node and set node online
T815_C_OFFLINE	remove node from net
T815_C_DIAG	return diagnostic value
T815_C_MAP	build a network map
T815_C_FLUSH	flush receive buffer
T815_C_TXSTATUS	return state of the last write request

See behind for more detailed information on each control code.

RETURNS

ioctl returns 0 if successful, or –1 on error.

On error, *errno* will contain a standard read error code (see also LynxOS System Call – read) or a special error code (see also below Additional Error Codes)

SEE ALSO

LynxOS System Call - ioctl().

3.5.1 T815_C_ONLINE

NAME

T815_C_ONLINE – Configure node end set node online

DESCRIPTION

With this ioctl function the selected node will be setup and set online to the connected arcnet.

A pointer to the configuration structure (*T815_CONFIG_BUF*) is passed by the parameter *arg* to the driver.

The *T815_CONFIG_BUF* structure has the following layout:

```
typedef struct
{
        unsigned long       io_flags;
        unsigned char       node_id;          /* used if io_flag 'NODE_ID' is set          */
        unsigned char       network_timeout;  /* this parameter controlls the Response,    */
                                              /* idle and Recon Times of our node in [us]  */
                                              /* Value     Resp    Idle     Reconfig       */
                                              /*   0       1130    1237      1680          */
                                              /*   1        563     624      1680          */
                                              /*   2        285     316      1680          */
                                              /*   3         78      86       840          */
        unsigned char       speed;            /* this parameter determine the network speed*/
                                              /*          Value            Network Speed   */
                                              /*  TP815_5000KBTS    5        Mbps          */
                                              /*  TP815_2500KBTS    2.5      Mbps          */
                                              /*  TP815_1250KBTS    1.25     Mbps          */
                                              /*  TP815_625KBTS     625      Kbps          */
                                              /*  TP815_312KBTS     312.5    Kbps          */
} T815_CONFIG_BUF, *PT815_CONFIG_BUF;
```

io_flags

This parameter is an ORed value of following flags:

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_BUF*. If not set, determine the node ID by reading the hardware DIP switch.

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.

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 following tables:

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

speed

Determines the network speed by setting the clock prescaler to one of five possible values:

Name	Network Speed	Moduletypes
T815_312KBTS	312.5 Kbps	-10 / -11 / -20 / -21
T815_625KBTS	625 Kbps	-10 / -11 / -20 / -21
T815_1250KBTS	1.25 Mbps	-10 / -11 / -20 / -21
T815_2500KBTS	2.5 Mbps	-10 / -11 / -20 / -21
T815_5000KBTS	5 Mbps	-11 / -21

EXAMPLE

```
int                fd;
int                result;
T815_CONF_BUF      cfgBuf;

/* Set node online (node: 5, 1.25 MBit, accept long packages,          */
/* Response Time: 37.35 ms (TIP815-21)                                 */

cfgBuf.io_flags         = T815_NODE_ID | T815_LONGPACKET;
cfgBuf.node_id          = 5;
cfgBuf.speed            = T815_1250KBTS;
cfgBuf.network_timeout  = 3;

result = ioctl(fd, T815_C_ONLINE, (char*)&cfgBuf);
if (result != OK)
{
     // process error;
}
```

3.5.2 T815_C_OFFLINE

NAME

T815_C_OFFLINE – Removes the node from the net

DESCRIPTION

With this ioctl function the selected node will be removed from the net and set offline.

No parameter needed, the parameter *arg* must be set to *NULL*

EXAMPLE

```
int                fd;
int                result;

result = ioctl(fd, T815_C_OFFLINE, NULL);
if (result != OK)
{
     // process error;
}
```

3.5.3 T815_C_DIAG

NAME

T815_C_DIAG – Read the number node reconfigurations

DESCRIPTION

This ioctl function will return the number of node reconfiguration occurred since the last read or start of initialization of the device.

A pointer to an unsigned long value is passed by the parameter *arg* to the driver, where the number of reconfigurations will be returned.

EXAMPLE

```
int                fd;
int                result;
unsigned long      numRecon;

result = ioctl(fd, T815_C_DIAG, (char*)&numRecon);
if (result != OK)
{
     // process error;
}
```

3.5.4 T815_C_MAP

NAME

T815_C_MAP – Build a network map

DESCRIPTION

With this ioctl function a network map can be build, which identifies active nodes on the net.

A pointer to an array of 32 byte is passed by the parameter *arg* to the driver. Nodes which are set online will be marked with a set bit in the array. The node assignment look like the following.

The first byte of the array (index 0) returns the states of the nodes 0 to 7. Bit 0 represents node 0, node 1 can be checked with bit 1 and so on. The second byte of the array (index 1) returns the states of node 8 to 15 and so on.

Example:

map[0] = 0x01	node 0 is online, nodes 1 .. 7 are offline
map[1] = 0x03	nodes 8, 9 are online, nodes 10 .. 15 are offline
map[2] = 0x80	node 23 is online, nodes 16 .. 22 are offline
map[3] = 0x00	nodes 24 .. 31 are offline
...	...
map[31] = 0x00	nodes 248 .. 255 are offline

EXAMPLE

```
int                fd;
int                result;
unsigned char      map[32];

result = ioctl(fd, T815_C_OFFLINE, (char*)map[0]);
if (result != OK)
{
     // process error;
}
```

3.5.5 T815_C_FLUSH

NAME

T815_C_FLUSH – Flush the receive buffer

DESCRIPTION

With this ioctl function the selected node will flush its receive buffer.

No parameter needed, the parameter *arg* must be set to *NULL*

EXAMPLE

```
int                fd;
int                result;

result = ioctl(fd, T815_C_FLUSH, NULL);
if (result != OK)
{
     // process error;
}
```

3.5.6 T815_C_TXSTATUS

NAME

T815_C_TXSTATUS – Returns the state of the last write command

DESCRIPTION

With this ioctl function returns the state of the last write command. This command is useful for messages that are written with the *T815_NOWAIT* option. The state is returned, using the standard error handling.

No parameter needed, the parameter *arg* must be set to *NULL*

EXAMPLE

```
int                fd;
int                result;

result = ioctl(fd, T815_C_TXSTATUS, NULL);
if (result != OK)
{
     // process error;
}
```

4 Debugging and Diagnostic

If your installed IPAC port driver (e.g. tip815) doesn't find any devices although the IPAC is properly plugged on a carrier port, it's interesting to know what's going on in the system.

Usually all TEWS TECHNOLOGIES device drivers announce significant events or errors via the device driver routine kkprintf(). To enable the debug output you must define the macro DEBUG in the device driver source files (e.g. carrier_class.c, carrier_tews_pci.c, tip815.c, ...).

The debug output should appear on the console. If not please check the symbol KKPF_PORT in uparam.h. This symbol should be configured to a valid COM port (e.g. SKDB_COM1).

The following output appears at the LynxOS debug console if the carrier and IPAC driver starts:

```
TEWS TECHNOLOGIES - IPAC Carrier Class Driver version 1.0.1 (2003-12-18)
IPAC_CC  :  IPAC (Manuf-ID=B3, Model#=12) recognized @ slot=1 carrier=<TEWS TEC>
TIP815 - ARCNET version 2.0.0 (2004-11-18)
TIP815  :  Probe new TIP815 mounted on <TEWS TECHNOLOGIES - VME Carrier> at slot B
```

If you can't solve the problem by yourself, please contact TEWS TECHNOLOGIES with a detailed description of the error condition, your system configuration and the debug outputs.

5 Additional Error Codes

Error	Value	Description
ENOTREADY	801	node is not online
ENOTACKN	802	No ACK from receiver after transmission
EBADPACKETSIZE	803	illegal packet size
EBADSPEED	804	invalid bit-rate selected
EQFULL	805	The Queue for incoming messages is full
EDUPID	806	illegal (duplicate) node id