



# VxWorks SENS Driver

*with Embedded RAINlink Technology*

(VxWorks 5.3.1/Tornado I; Sun Ultrasparc CP1500 BSP 1.1/4)

## Installation and User's Guide

### Overview

The VxWorks 5.3.1 driver for the ZNYX Networks NetBlaster™ CompactPCI network interface includes embedded RAINlink technology. RAINlink provides critical link services such as link aggregation, link failover, and dynamic load balancing over multiple network ports. These ports can be a group of single-channel or multi-channel ZNYX Networks adapter ports, up to a maximum of 32 ports.

The VxWorks 5.3.1 driver consists of an object file and configuration source code and header files. The appropriate driver can be obtained from the ZNYX Networks web site: <http://www.znyx.com>. The distribution is available as a self-extracting archive. The installation consists of placing files in the correct build directories, and tailoring the build environment to include the ZNYX Networks driver in the VxWorks 5.3.1 image.

These instructions install the ZNYX Networks network driver to bring up individual ports without RAINlink features enabled. Examples are included to bring up the most common RAINlink configurations for four ports: A system-to-system trunk, a Fast EtherChannel system-to-switch trunk, and a failover group. All the instructions are duplicated in the driver distribution readme.txt.

### Systems Requirements

<b>Platforms:</b>	Sun CP1500-based CompactPCI System
<b>Operating Systems:</b>	VxWorks 5.3.1 (Tornado I) CP1500 BSP 1.1/4

## Supported ZNYX Networks NetBlaster Adapters (CP1500):

Model	Bus	Number of Ports
442 hot-swap	cPCI	2
444 hot-swap	cPCI	4
474 rear i/o & hot-swap	cPCI	4
478 rear i/o & hot-swap	cPCI	8

### Retrieving the ZNYX Networks Drivers

1. Go to the ZNYX Networks web site at <http://www.znyx.com> and navigate under the driver downloads area to get the VxWorks 5.3.1 driver for your adapter. Be sure to choose the driver for CP1500 UltraSparc platform.
2. The driver is available as a self-extracting archive.

### Modifications to the CP1500 BSP

The CP1500 BSP provided as part of VxWorks 5.3.1 provides very limited and rudimentary PCI support. It provides no support for the 21152 bridges used by our ZX210 and ZX410 series products, and therefore this driver only supports the ZX440 and ZX470 adapters which use non-transparent bridges.

Modifications need to be made to the BSP to provide needed support for the included `if_zxePci.c`. These modifications are targeted to BSP revision 1.1/4. Later BSP versions may need different modifications. If the PCI bus/device addresses are changed, the modifications will need to be updated to track them.

1. Modify the file `$(WIND_BASE)/target/config/cp1500/config.h`:

#### A. Add

```
#define PCI_BUS_A_MEM_ADRS_ZNYX      0x90000000
```

between

```
#define PCI_BUS_CONFIG_SIZE      0x01000000  
#define PCI_BUS_A_MEM_ADRS      0xd0000000
```

#### B. Below `#define ZT6660_INT_NUM...` add:

```
#define ZX_INT_NUM      11
```

2. Modify the file: `$(WIND_BASE)/target/config/cp1500/memDesc.c`

Below or above the structure definition for "PCI memory space for Bus A" add the ZNYX Networks virtual mapping for Bus A. This MUST be added for the driver to work, as it cannot use the byte-swapped virtual mapping provided by the standard BSP.

```
/*
```

```

* PCI memory space for bus A (unswapped - ZNYX addition)
*/
{
  (void *) PCI_BUS_A_MEM_ADRS_ZNYX,
  (void *) (PCI_BUS_A_MEM_PA >> PAGE_SHIFT),
  PCI_BUS_A_MEM_SIZE,
  VM_SPACE_MASK,
  VM_IO_BITS
},

```

3. Modify the file: \$(WIND\_BASE)/target/config/cp1500/sysLib.c

A. Below the first `#ifdef INCLUDE_ZT6660... #endif` block (line ~139), add:

```

#ifdef INCLUDE_ZXE
/*
 * This is predicated on an idea that all devices will be on
 * (bus/dev) 3/12 to 3/15. zxIno[dev - 12] != -1 means a device
 * actually exists.
 *
 * It is assumed that all boards will share the same SW interrupt
 */

#define ZXE_MAX_BOARDS 4
#define ZXE_PCI_BUS 3
#define ZXE_LOWEST_DEV 12

int  zxIvec[ZXE_MAX_BOARDS];      /* ZXE INO */
int  zxIno[ZXE_MAX_BOARDS];      /* ZXE INO */

#endif      /* INCLUDE_ZXE */

```

B. Add "int i;" to `sysIntAck`, and then below the `#ifdef INCLUDE_ZT6660` block (line ~627) add

```

#ifdef INCLUDE_ZXE
  case INT_TO_TRAPNUM(ZX_INT_NUM):
    sysSoftIntClear (1 << ZX_INT_NUM);
    for (i = 0; i < ZXE_MAX_BOARDS; i++) {
      if (zxIno[i] != -1) { sysClearUpaInt(zxIno[i]); }
    }
    break;
#endif      /* INCLUDE_ZNYX */

```

C. In `sysIntInit`, add "int i" as a variable and then below the `INCLUDE_ZT6660` block (approx line 830) add:

```

#ifdef INCLUDE_ZXE

  for (i = 0; i < 4; i++) {
    if (zxIno[i] != -1) {
      zxIno[i] = INR(0x1f, i);
    }
  }

  for (i = 0; i < ZXE_MAX_BOARDS; i++) {

```

```

        sysUpaIntMap [zxIno[i]] = ZX_INT_NUM;
        zxIvec[i] = INT_TO_TRAPNUM(ZX_INT_NUM);
    }

#endif

```

D. Before `sysPciBusConfig`, add this code fragment. Each BAR must be at least 8K in size.

```

#ifdef INCLUDE_ZXE
struct zxePciBarConfig {
    unsigned long bar0, bar3;
};

struct zxePciBarConfig zxeBarConfig[] = {
    {0x41000000, 0x41800000},
    {0x40080000, 0x40200000},
    {0x43000000, 0x43800000},
    {0x44000000, 0x44800000},
};
#endif

```

E. Add these lines to the beginning of `sysPciBusConfig`:

```

USHORT svend_id;
ULONG reset_var;
int i;

```

F. PCI bus mastering arbitration for the A-Bus is not enabled in the current BSP version. In `sysPciBusConfig` (line ~1065) above `/* Enable arbitration on APB B-Bus for all slots. */` add this code fragment:

```

#if 1
    /* Enable arbitration on APB A-Bus for all slots. */
    pciConfigInLong (0, 1, 0, 0xe0, &ctrl);
    ctrl |= 0xf;
    pciConfigOutLong (0, 1, 0, 0xe0, ctrl);
#endif

```

G. Further in the same function, after the `#ifdef INCLUDE_ZT6660` code block, add:

```

#ifdef INCLUDE_ZXE

/* This portion just scans for our device, and
 *sets zxIno[i] to -1 or 0
 */

/* Actual HW address setting is in if_zxePci.c */

for (i = 0; i < ZXE_MAX_BOARDS; i++) {
    pciConfigInWord(ZXE_PCI_BUS, ZXE_LOWEST_DEV + i,
        0, 44, &svend_id);
    if (svend_id == 0x110d) {

        /* Assert chip reset and wait for it to come back */

        /* 0xD8 is the chip reset register. */

```

```

pciConfigOutLong(ZXE_PCI_BUS, ZXE_LOWEST_DEV + i,
 0, 0xd8, 2);

reset_var = 0x02;
while (reset_var && 0x2) pciConfigInLong(ZXE_PCI_BUS,
ZXE_LOWEST_DEV+i, 0, 0xd8, &reset_var);

zxIno[i] = 0;
/* Program I/O and Memory base addresses */
pciConfigOutLong (ZXE_PCI_BUS, ZXE_LOWEST_DEV+i, 0,
PCI_CFG_BASE_ADDRESS_0, zxeBarConfig[i].bar0);
pciConfigOutLong (ZXE_PCI_BUS, ZXE_LOWEST_DEV+i, 0,
PCI_CFG_BASE_ADDRESS_3, zxeBarConfig[i].bar3);
/* Enable I/O and Memory space */
pciConfigOutWord (ZXE_PCI_BUS, ZXE_LOWEST_DEV+i, 0,
PCI_CFG_COMMAND, PCI_CMD_MEM_ENABLE |
PCI_CMD_IO_ENABLE);
/* Clear status register */
pciConfigOutWord (ZXE_PCI_BUS, i+ZXE_LOWEST_DEV, 0,
PCI_CFG_STATUS, 0xffff);
} else {
zxIno[i] = -1;
}
}
#endif

```

## ***Installing the ZNYX Networks Driver for VxWorks 5.3.1***

1. Copy the following files into the `$(WIND_BASE)/target/config/$(BSP)` directory

```

if_zxe.obj /* driver obj file */

if_zxePci.c /* Znyx PCI Library */
if_zxePci.h

if_zxeRlk.c /* Znyx RAINLink configuration */
if_zxeRlk.h

configZxe.c /* Znyx startup routines(optional) */
configZxe.h

```

2. Change to the `$(WIND_BASE)/target/config/$(BSP)` directory. Edit the Makefile. Add "if\_zxe.obj", "configZxe.o", "if\_zxeRlk.o" and "if\_zxePci.o" to the MACH\_EXTRA depending on your boot configuration. The list should look similar to the following when you are finished:

```
MACH_EXTRA = if_zxe.obj configZxe.o if_zxeRlk.o if_zxePci.o
```

3. Modify the file `$(WIND_BASE)/target/config/$(BSP)/config.h` to add the following definitions (near `#define INCLUDE_PCI`):

(change current definition of)

```

#define IP_MAX_UNITS      8          /* modify for your configuration
*/

#define INCLUDE_ZXE          /* include ZNYX support */

#ifndef STANDALONE_NET
#define STANDALONE_NET
#endif

```

4. (Optional) Modify the file `$(WIND_BASE)/target/config/$(BSP)/configNet.h`. After the entry:

```

{ 0, END_TBL_END, NULL, 0, NULL, FALSE},
};

```

Add:

```

#ifdef INCLUDE_ZXE
#include "configZxe.h" /* ZNYX definition of zxeDevTbl[] */
#endif

```

5. Rebuild the `vxWorks.st` file. In the `$(WIND_BASE)/target/config/cp1500` directory execute:

```
make vxWorks.st
```

This command is case sensitive.

6. The `vxWorks.st` standalone image can now be booted via the network, or off a floppy, etc. Refer to `$(WIND_BASE)/target/config/$(BSP)/target.txt` for details of the available boot methods.

The file `configZxe.c` contains a sample routine, `zxeUp()`, that loads and starts the driver for a specified device using the default configuration. There is also a `zxeDown()` routine to bring down the interface. These routines are sometimes handy during development to bring the devices up manually, after the image has booted. Normally, all devices will be built into the VxWorks image, and the driver will load upon boot. Use the routines as an example of the functions necessary to bring up an interface.

7. If `configZxe.c` is included in the kernel, issue the following command from the console after VxWorks has booted to load and start the driver:

```
zxeUp "device_name", "IP Address", netmask, media_type, instance
```

Note: the first device is usually occupied by the onboard ethernet, so choose the first available device. For example, to load and start the first instance of the driver, enter:

```
zxeUp "znb1", "192.168.116.204", 0xfffffe00, 0, 0
```

To load a RAINlink device, use `zrl` instead of `znb`. The last parameter indicates the index into the RAINlink configuration table. For example, if you wished to bring up a Fast Failover group with two ports, you would provide the index of 5. Enter:

```
zxeUp "zr11", "192.168.116.204", 0xfffffe00, 0, 5
```

## Configuring RAINlink for VxWorks 5.3.1

RAINlink features are statically added into your VxWorks image at compile time. The RAIN management configuration table in `if_zxeRlk.c` contains the available RAINlink configurations. ZNYX Networks provides six standard configurations for basic two-port and four-port trunking and failover groups. The table below shows the provided RAINlink configurations, the name of the table entry, and the corresponding index number:

Configuration	Name	Index Number
System-to-System trunking between 4 ports	<code>zxe_rlss4</code>	0
Fast EtherChannel trunking between 4 ports	<code>zxe_rlfe4</code>	1
Fast Failover between 4 ports	<code>zxe_rlff4</code>	2
System-to-System trunking between 2 ports	<code>zxe_rlss2</code>	3
Fast EtherChannel trunking between 2 ports	<code>zxe_rlfe2</code>	4
Fast Failover between 2 ports	<code>zxe_rlff2</code>	5

Standard single port ZNYX Networks devices have the name `znb`. RAINlink-configured devices have the name `zrl`. To use one of the provided configurations, create an END table entry for the RAINlink configuration you wish to load, and then load the driver. Examine `configZxe.c` and implement the method that best fits your applications needs to bring up the network interfaces. Reference `usrNetwork.c`, and `bootConfig.c` for routines to bring up the interface. A typical entry looks like:

```
{ 0, ZRL_LOAD_FUNC, "0:0:0:0:0:0:0", ZXE_BUFF_LOAN, NULL, FALSE },
```

The number at the beginning of the line specifies the unit number for the device. The unit number in the line above is 0, therefore the device name it loads is `zrl0`. `ZRL_LOAD_FUNC` is the name of the load function for a RAINlink device. Load functions for `znb` and `zrl` devices are:

```
#define ZNB_LOAD_FUNC      znbEndLoad /*driver external interface*/
#define ZRL_LOAD_FUNC      zrlEndLoad /*driver external interface*/
```

The next element is the init string, which contains all the device-specific parameters. For `zrl` devices, the first field of the init string is the RAINlink configuration table index (`zxe_rlk_config_tbl[]` in `if_zxeRlk.c`). For `znb` devices, the first field refers to the `ppa` of that device. The init string format consists of:

```
ppa|index:memBase:mediaType:mtu:recvBufs:xmitBufs:staticBufs:flags
```

where:

Name	Description
<code>&lt;ppa index&gt;</code>	specifies the <code>ppa</code> or configuration record to use
<code>&lt;memBase&gt;</code>	defines main memory base as seen from PCI bus
<code>&lt;mediaType&gt;</code>	defines the media type of the device
<code>&lt;recvBufs&gt;</code>	defines the number of receive buffers per <code>ppa</code>
<code>&lt;xmitBufs&gt;</code>	defines the number of transmit buffers per <code>ppa</code>
<code>&lt;staticBufs&gt;</code>	defines the number of static buffers per <code>ppa</code>
<code>&lt;flags&gt;</code>	defines various interface flags

The next element in the table entry defines buffer loaning:

```
#define ZXE_BUFF_LOAN 1 /* enable buffer loaning */
```

The last two elements are a NULL reserved for the BSP, and a TRUE/FALSE indicator of whether or not the load request for that device has been processed.

For example, an END table entry for a zrl device such as a 4-port Fast EtherChannel, Layer 3 trunk, would be the following:

```
{ 0, ZRL_LOAD_FUNC, "1:0:0:0:0:0:0", ZXE_BUFF_LOAN, NULL, FALSE },
```

If you wish to customize configuration for trunks for other than two or four ports, or disable the dynamic load balancing, go to the "Advanced Trunking" section. If you wish to configure failover groups for other than two or four ports, or set timeout parameters, then go to the "Advanced Failover" section. You can also configure failover groups of trunks themselves. Always configure trunks before failover groups.

## Advanced Trunking

To add a new trunk configuration, edit the file `if_zxeRlk.c` and add an entry for the trunk. Each entry corresponds to one device and contains the `RLK_TRUNK` type, the desired mode, a 0 for the timeout field, and -1 terminated list of ports to be included in that trunk. Acceptable modes for trunks are:

Mode:	Appropriate for:
<code>IP_TRUNKING_MODE</code>	System-to-system trunking
<code>LAYER_2_MODE</code>	System-to-switch trunking
<code>LAYER_3_MODE</code>	System-to-switch trunking
<code>NO_BALANCE_MODE</code>	System-to-switch trunking. Use a logical OR in the mode field to disable dynamic load balancing. (Layer 2 or Layer 3 only)

For system-to-system trunking, use `IP_TRUNKING_MODE`. Both systems should be setup identically. In `IP_TRUNKING_MODE`, packets are sent to the driver as large datagrams and fragmented across the active links.

For system-to-switch trunking, the default method of load balancing scheme uses Layer 3 Protocols. Layer 3 mode balances traffic across the available ports based on IP addresses, as opposed to Layer 2, which is based on MAC addresses. If you wish to use only Layer 2 protocols for load balancing, change the second element to `LAYER_2_MODE`.

For system-to-switch trunking, you can disable dynamic load balancing, which is on by default. Dynamic load balancing periodically tries to balance the load across the available members of the trunk, and is normally left enabled. Disable it by performing a logical OR of `NO_BALANCE_MODE` and either `LAYER_2_MODE` or `LAYER_3_MODE` in the mode field.

You can use any available ports in any order. Port entries need to be in order, and should not be duplicated in trunks. A trunk must consist of at least two ports, and cannot contain other trunks. You can have more than one trunk. Each trunk will have its own entry in the device table. For example, to configure two system-to-switch trunks of four ports each, use layer 3 mode, and disable dynamic load balancing, you would add two table element definitions in `if_zxeRlk.c`:

```
/*
 * Two 4-Port Fast EtherChannel Trunks, layer 3 mode with
 * balance mode disabled.
 */
zxe_rlk_ppa_list_t zxe_rlfe4a[] = {
    { RLK_TRUNK, LAYER_3_MODE | NO_BALANCE_MODE, 0,
      { 0, 1, 2, 3, -1 } },
    { 0, }
};
zxe_rlk_ppa_list_t zxe_rlfe4b[] = {
    { RLK_TRUNK, LAYER_3_MODE | NO_BALANCE_MODE, 0,
      { 4, 5, 6, 7, -1 } },
    { 0, }
};
```

You would also need to add the entries to the `zxe_rlk_config_tbl[]` in `if_zxeRlk.c`:

```
zxe_rlk_ppa_list_t *zxe_rlk_config_tbl[] = {
    zxe_rlss4,          /* 0 */
    zxe_rlfe4,          /* 1 */
    zxe_rlff4,          /* 2 */
    zxe_rlss2,          /* 3 */
    zxe_rlfe2,          /* 4 */
    zxe_rlff2,          /* 5 */
    zxe_rlfe4a,         /* 6 new first trunk */
    zxe_rlfe4b,         /* 7 new second trunk */
    0
};
```

Add two END table entries for `zrl0` and `zrl1`, and bring up the network interfaces.

Examine `configZxe.c` and implement the method that best fits your applications needs to bring up the network interfaces.

### **Advanced Failover**

You can create link failover groups, assign ports and trunks to these groups and select a modes of failover. Failover groups are configured in the same manner as trunks: Edit the `if_zxeRlk.c` file and rebuild the kernel.

Acceptable modes for failover groups are:

Mode:	Appropriate for:
<code>timeout_mode &lt;time&gt;</code>	Failover mode; time in milliseconds

Failover groups can contain individual ports, trunks, or both ports and trunks. Always configure trunks before failover groups. Add an entry for each failover group containing the list of ports and trunks to be included in that group. The default mode of failover is fast failover. In this mode, RAINlink moves the traffic over to a redundant stand-by link in case of a link failure in as little as 500 milliseconds.

In addition to fast failover, timeout mode can be enabled for a failover group. By enabling timeout mode, if no traffic is received within the specified time interval, the active port is automatically switched to another available link. You can choose different failover modes for different groups. You can also specify different timeout intervals. To build a failover group of two trunks, first build the trunks, then the failover group and set the timeout mode on with a value of 30 seconds:

```
/*
 * 2-Port Failover of 2 2-Port Trunks, timeout=30 seconds
 */
zxe_rlk_ppa_list_t zxe_rlff2ss2[] = {
    { RLK_TRUNK, IP_TRUNKING_MODE, 0, { 0, 1, -1 } },
    { RLK_TRUNK, IP_TRUNKING_MODE, 0, { 2, 3, -1 } },
    { RLK_FAILOVER, TIMEOUT_MODE, 30000, { 0, 2, -1 } },
    { 0, }
};
```

This results in one RAINlink device. You would also need to add the entries to the `zxe_rlk_config_tbl[]` in `if_zxeRlk.c`:

```
zxe_rlk_ppa_list_t *zxe_rlk_config_tbl[] = {
    zxe_rlss4,          /* 0 */
    zxe_rlfe4,          /* 1 */
    zxe_rlff4,          /* 2 */
    zxe_rlss2,          /* 3 */
    zxe_rlfe2,          /* 4 */
    zxe_rlff2,          /* 5 */
    zxe_rlff2ss2,      /* 6 new failover */
    0
};
```

Add an END table entry for zrl0, and bring up the network interface.

Examine configZxe.c and implement the method that best fits your applications needs to bring up the network interfaces.

## ZNYX Networks VxWorks Utilities and Sample Routines

RAINlink includes the following files and utilities to aid in configuration and monitoring:

Name	Description
zxeUp	Initializes the driver and the device
znbEndLoad	Initialize the NetBlaster znb driver and device
zrlEndLoad	Initialize the RAINlink zrl driver and device
zxeVersion	Displays the driver version
rconfigShow	Displays the current RAINlink configuration
rlportstateShow	Displays the current port status/statistics
zxeDown	Brings down the driver
zxeAllMultiSet	Set Promiscuous Multicast Mode
ZxeAllMultiClear	Clears Promiscuous Multicast Mode

This section includes Unix-style man pages to be used as reference. The file `zxeConfig.c` contains the sample routine, `zxeUp()`, that loads and starts the `zxe` driver for a specified device using the default configuration and a sample routine, `zxeDown()`, that stops and unloads the `zxe` driver for that device. To include in the kernel, copy the file to the `$(WIND_BASE)/target/config/$(BSP)` directory and include it in `sysLib.c`

### zxeUp()

<b>NAME</b>	<b>zxeUp()</b> – Initializes the driver and the device.
<b>SYNOPSIS</b>	<pre> int zxeUp(     char *devString,           /* device to initialize, e.g. */                               /* "znb0", "zrl" */     char *pAddr,              /* IP address of the interface */     ULONG iMask,              /* IP netmask of the interface */     int mediaType,            /* media type, 0-7 */     int ppa   index           /* Specifies the ppa or configuration */                               /* index to use for RAINlink */ ) </pre>
<b>DESCRIPTION</b>	<p>This routine initializes the driver and the device to the operational state.</p> <p>The following device strings are acceptable:</p> <pre> "znb<i>n</i>"           /* NetBlaster device <i>n</i>=0-31*/ "zrl<i>n</i>"           /* RAINlink device <i>n</i>=0-31*/ </pre> <p>The following media types are defined:</p> <pre> MEDIA_AUTO       0    /* auto negotiation */ MEDIA_TP         1    /* 10 Mbit - half duplex */ MEDIA_BNC        2    /* not supported */ MEDIA_AUI        3    /* not supported */ MEDIA_TPFD       4    /* 10 Mbit - full duplex */ MEDIA_TX         5    /* 100 Mbit - half duplex */ MEDIA_TXFD       6    /* 100 Mbit - full duplex */ MEDIA_T4         7    /* not supported */ </pre>
<b>RETURNS</b>	OK or ERROR

## **znbEndLoad() zrlEndLoad()**

**NAME** **znbEndLoad, zrlEndLoad** - initialize the driver and device

**SYNOPSIS**

```
END_OBJ * znbEndLoad (  
  char *initString, /* device specific parameters */  
  void *pBsp /* not used by the driver */  
  )  
END_OBJ * zrlEndLoad (  
  char *initString, /* device specific parameters */  
  void *pBsp /* not used by the driver */  
  )
```

**DESCRIPTION** Initializes the driver and the device to the operational state. All of the device specific parameters are passed in the *initString*. The void pointer parameter is currently not used by the driver.

*initstring* format:

*unit:ppa | index:memBase:mediaType:rcvBufs:xmitBufs:staticBufs:flags*

<i>unit</i>	- The device unit to initialize
<i>ppa   index</i>	- The <i>ppa</i> or configuration record to use.
<i>memBase</i>	- Main memory base as seen from PCI bus.
<i>mediaType</i>	- Media type of the device.
<i>rcvBufs</i>	- The number of receive buffers per <i>ppa</i>
<i>xmitBufs</i>	- The number of transmit buffers per <i>ppa</i>
<i>staticBufs</i>	- The number of static buffers per <i>ppa</i>
<i>flags</i>	- Defines various interface flags.

The following *mediaTypes* are defined:

<b>MEDIA_AUTO</b>	<b>0</b>	<b>/* auto negotiation */</b>
<b>MEDIA_TP</b>	<b>1</b>	<b>/* 10 Mbit - half duplex */</b>
<b>MEDIA_BNC</b>	<b>2</b>	<b>/* not supported */</b>
<b>MEDIA_AUI</b>	<b>3</b>	<b>/* not supported */</b>
<b>MEDIA_TPFD</b>	<b>4</b>	<b>/* 10 Mbit - full duplex */</b>
<b>MEDIA_TX</b>	<b>5</b>	<b>/* 100 Mbit - half duplex */</b>
<b>MEDIA_TXFD</b>	<b>6</b>	<b>/* 100 Mbit - full duplex */</b>
<b>MEDIA_T4</b>	<b>7</b>	<b>/* not supported */</b>

The following *flags* are defined:

**END\_NOCOPY\_ON\_TRANSMIT**    **0x100**

**RETURNS** An END object pointer or NULL on error.

### **zxeVersion()**

---

**NAME** **zxeVersion()** – returns the version of the znb driver

**SYNOPSIS** **int zxeVersion()**

**DESCRIPTION** Returns the version of the driver

**RETURNS** OK or ERROR

### **rlportstateShow()**

---

**NAME** **rlportstateShow()** – Displays current port statistics

**SYNOPSIS** **int rlportstateShow(**  
**int port /\* port == device unit \*/**  
**)**

**DESCRIPTION** This routine displays the current status and statistics for the designated *port*. *port* refers to the specific hardware port, not to a zrl|znb unit.

**RETURNS** OK or ERROR

### **rlconfigShow()**

---

**NAME** **rlconfigShow()** – Displays current RAINlink configuration

**SYNOPSIS** **int rlconfigShow()**

**DESCRIPTION** This routine displays the current RAINlink configuration

**RETURNS** OK or ERROR

### **zxeDown()**

---

**NAME** **zxeDown()** - Stop and unload an instance of the driver.

**SYNOPSIS** **int zxeDown(**  
**int unit /\* device unit, 0-31 \*/**  
**)**

**DESCRIPTION** This routine brings down an interface and stops and unloads the driver and device for that interface.

**RETURNS** OK or ERROR

### **zxeAllMultiSet()**

---

**NAME** **zxeAllMultiSet()** - Set Promiscuous Multicast on selected unit

**SYNOPSIS** **int zxeAllMultiSet(**  
**char \*name /\* device name \*/**  
**)**

**DESCRIPTION** This routine sets promiscuous multicast on the selected unit. String format is "znb(x)" or "zrl(x)" e.g. "znb1", "zrl12", etc. Strict string checking is performed against the name parameter.

**RETURNS** OK or ERROR

## **zxeAllMultiClear ( )**

---

<b>NAME</b>	<b>zxeAllMultiClear ( )</b> - Clears Promiscuous Multicast on selected unit
<b>SYNOPSIS</b>	<b>int zxeAllMultiClear( char *name /* device name */ )</b>
<b>DESCRIPTION</b>	This routine clears promiscuous multicast on the selected unit. String format is "znb(x)" or "zrl(x)" e.g. "znb1", "zrl12", etc. Strict string checking is performed against the name parameter.
<b>RETURNS</b>	OK or ERROR

### ***Changes Since Last Release***

- Added routines to set and clear promiscuous multicast.
- Cleaned up documentation for use of zxeUp() in source files.

### ***Discrepancies***

- RAINlink trunks with invalid ports are not handled correctly. Properly configured systems should not be affected.
- The “down” routine does not return all resources. A report has been filed with Wind River Systems, and they are working on a solution to the problem.

## ***Where to go for Technical Support***

<b>Resource</b>	<b>Address</b>
Telephone	(510) 249-0800
Toll-Free	(800) 724-0911 (USA Only)
FAX	(510) 656-2460
Website	<a href="http://www.znyx.com">www.znyx.com</a>
E-mail	<a href="mailto:support@znyx.com">support@znyx.com</a>



ZNYX Networks  
48501 Warm Springs Blvd., Suite 107  
Fremont, CA 94539 USA

Tel.: (510) 249-0800  
Toll-Free: (800) 724-0911  
Fax: (510) 656-2460  
E-mail: [sales@znyx.com](mailto:sales@znyx.com)  
Website: [www.znyx.com](http://www.znyx.com)

Document # DC0119-03  
Version # 25-April-2000

© 2000 ZNYX Networks. All rights reserved worldwide. Information in this document is subject to change without prior notice. ZNYX, RAIN, RAINlink, RAINlink for VxWorks, and NetBlaster are trademarks or registered trademarks of ZNYX Networks Corporation in the United States and/or other countries. All other marks, trademarks or service marks are the property of their respective owners.

ZNYX Networks may have patents, pending patent applications, trademarks, copyrights or other intellectual property rights covered in the subject matter of this document. By furnishing this document, ZNYX Networks does not license nor waive its license to those intellectual property rights except as expressly provided in a written license agreement from ZNYX Networks. Information in this document is subject to change without prior notice.