ZD1201 802.11b Wireless LAN Card HOWTO (USB)

1. Introduction:

Because more and more people install the Linux operating system in the desktop and notebook, we provide the Linux solution of our ZD1201 802.11b Wireless LAN Card. This document is intended to describe how to setup and how to use ZD1201 under the Linux operating system. It is supposed that you have idea of some basic concepts about IEEE 802.11 such as what is AP (Access Point), what is SSID, etc.

We modify the source code written by Absolute Value System to be our device driver on the linux platform. It is an open source package licensed by GPL. One can download this code from http://www.linux-wlan.com/ freely. In this package, it provides four interfaces including PCMCIA, PCI, PLX and USB. Now we provide the PCMCIA, PCI and USB drivers for ZD1201 wireless card. In this document, we focus on the USB driver. For PCMCIA and PCI interfaces, one can reference another HOWTO.

2. Building the device driver:

In this section, we will describe how to build our device driver under the Linux operating system.

1.Uncompress the package:

tar zxvf zdsta-date.tar.gz (where date is the released date of our package, such as 3617)

The first thing one should do is uncompress this package by tar. After untar this package, it will create a directory named zdsta-x.y.z. The value x, y, z represent the major and minor version number of this package. One should change directory into this directory.

2.Kernel Checking:

Make sure you have configured kernel and (optionally) pcmcia sources on your system. Note that if you are only building the prism2_pci, drivers you don't need the pcmcia-cs source tree. If fact, there are lots of things one should know when building the pcmcia driver. One can reference it on the Linux PCMCIA HOWTO which can be downloaded from http://pcmcia-cs.sourceforge.net/

3.Configuration:

To configure the linux-wlan-ng package, run 'make config'. You may be prompted by the following questions. The default answer is in braces [x]. If the default answer is the same with your system configuration, you can just press the <Enter> to select the default answer.

- Build Prism2.5 USB (_usb) driver? (y/n) [y]:
  Select "y" if you want to build the USB driver.

- Linux source directory [/usr/src/linux-2.4]:
  You should assign the Linux kernel source tree here. If the path is wrong, you will be prompted with an error message. Besides, you may get the following warning messages after answering this question.

```
WARNING: .config not present in kernel source tree. This will
         screw up modversions detection and build optimizations.
         Is this a RedHat kernel? Workarounds enabled.
         For an optimized build, copy over the relevent file from
         /usr/src/linux-2.4/configs/ to /usr/src/linux-2.4/.config
```

When you are prompted with this message, it represents that there is no “*.config* “ file in your

Linux kernel source directory. One can solve this problem by the following methods. One is to copy the relevant configuration file under the /usr/src/linux-2.4/configs/ to /usr/src/linux-2.4/.config (In my system, the Linux kernel directory is /usr/src/linux-2.4/). Another is change directory to the linux kernel directory and type "*make config*" or "*make menuconfig*" to generate the .config file. Under the Xwindows environment (including GNOME and KDE), you can type "*make xconfig*" under the linux kernel directory to generate the .config file.

- Alternate target install root directory on host []:
  This prompt allows you to specify an alternative root directory for the install process.

- Module install directory [/lib/modules/2.4.2-2]:
  Select where you want the driver modules to be installed. Usually, you don't need to change the location. Because the configuration script will use kernel version got from 'uname –r' as the module install path.

- Prefix for build host compiler? (rarely needed) []:
  When cross-compiling or using different compilers for kernel and user-mode software, it is sometimes (but rarely) necessary to specify a different compiler prefix to use when compiling the tools that are built to run on the build host during the linux-wlan-ng build process.

4.Build the package:

Just type "*make all*" to make the device driver.

5. Install the package:

Just type "*make install*" to install the device driver to the module directory.

3. Getting Start:

In the linux-wlan package, the README under the zdsta-x.y.z directory describes a way to modifying the /etc/wlan/* to set some configuration and connect to the Access Point. We will describe another methods in this section.

3.1 wlanctl-ng:

This command is included in the linux-wlan package which provides the way for an user to issue command to the driver.

1. Loading module:
   First, you have to be sure that the device driver is load. One can use "*lsmod*" command to list the module loaded into kernel. If the module is not load, one can use "*modprobe"* to load the module. In our usb driver, our driver will be loaded automatically, if our device is detected by the operating system.

```
]$ modprobe prism2_usb
```

2. Firmware download:
   Our ZD1201 is firmware-based. One should download the firmware into the network interface card to make our NIC work. Generally speaking, if the prism2_usb module is not loaded manually, the firmware will be automatically loaded. Of course, one can reload the firmware by the following command.

```
]$ wlanctl-ng wlan0 lnxreq_ifstate ifstate=enable
```

3. Auto join command:

```
]$ wlanctl-ng wlan0 lnxreq_autojoin ssid=<SSID> authtype=Type
```

Type:

opensystem
sharedkey

The autojoin command can help user easily join an Access Point. It will handle all the authentication and association things.

4. Set Channel

]$ wlanctl-ng wlan0 dot11req_mibset mibattribute= p2CnfOwnChannel=Value

The Value should be in the range 1~14.

5. Set RTSThreshold:

]$ wlanctl-ng wlan0 dot11req_mibset mibattribute=dot11RTSThreshold=Value

The Value should be in the range 0~2347

6. Set FragmentThreshold:

]$ wlanctl-ng wlan0 dot11req_mibset mibattribute=dot11FragmentThreshold =Value

The Value should be in the range 256~2346

7. Set SSID

]$ wlanctl-ng wlan0 dot11req_mibset mibattribute= p2CnfOwnSSID=<your ssid>

8. Set WEP Key

]$ wlanctl-ng wlan0 dot11req_mibset mibattribute=dot11WEPDefaultKeyID=<KeyIndex>
]$ wlanctl-ng wlan0 dot11req_mibset mibattribute=dot11ExcludeUnencrypted=true
]$ wlanctl-ng wlan0 dot11req_mibset mibattribute=dot11PrivacyInvoked=true
]$ wlanctl-ng wlan0 dot11req_mibset mibattribute=dot11WEPDefaultKey<KeyIndex>=xx:xx:xx..

*KeyIndex should be in the range of 0~3
** xx:xx:xx… is the WEP key. xx is in the hexdecimal format. If you want to use WEP 64, 5 hexdecimal values are expected. 13 hexdecimal values are expected when using WEP 128.

The four commands listed above are the basic commands you have to issue when enabling WEP and setting WEP key. One can set four WEP keys. If you want to set four keys at the same time, you can repeat the last command with changing the KeyIndex.

After setting the WEP key information, one can join to an Access Point by the following autojoin command.

]$ wlanctl-ng wlan0 lnxreq_autojoin ssid=<SSID> authtype=sharedkey

9. Site Survey

]$ wlanctl-ng wlan0 dot11req_scan bsstype=any bssid=00:00:00:00:00:00 \
scantype=both probedelay=0 channellist="01:02:03:04:05:06:07:08:09:0a:0b:0c:0d:00”\
minchanneltime=200 maxchanneltime=1000 ssid=""

]$ wlanctl-ng wlan0 dot11req_mibget mibattribute=p2CommunicationScanresult

It includes two commands when doing site survey. The first command is issue a scanresult command to the firmware which asks firmware to scan the channel. The second one is display the scan

result returned by firmware.

Site survey is very useful when searching the APs. One may want to join the AP with the specific SSID or join the AP which has the best signal.

10. Get Tally information

]$ wlanctl-ng wlan0 dot11req_mibget mibattribute=p2CommunicationTallies

In our ZD1201 STA, we provide the following tally information.

Field name	Size (words)
TxUnicastFrames	2
TxMulticastFrames	2
TxFragmentCount	2
TxUnicastOctets	2
TxMulticastOctets	2
TxRetryCount	2
TxMultipleRetryCount	2
TxMultipleRetryFrames	2
TxRetryLimitExceeded	2
AckFailureCount	2
RxUnicastFrames	2
RxMulticastFrames	2
RxPLCPCRC16ErrCnt	2
RxUnicastOctets	2
RxMulticastOctets	2
RxCRC32ErrCnt	2
RxQFullDiscardCnt + RxMemoryFullCnt	2
RxTotalCnt	2
RxFilterDecryptFailCnt	2
RxFilterDuplicatedCnt	2
CLOCK32_REG	2
RxFragmentCount	2
RTSSuccessCount	2
RTSFailureCount	2

3.2 iwconfig:

The original linux-wlan package provides partial wireless extension interface. All it provides are GET commands. So we can't use iwconfig to associate with an Access Point and we can't change the WEP key by iwconfig. But we can query the information by the iwconfig. We add some wireless extension interfaces to make it possible for users to use iwconfig to set WEP keys and associate with an Access Point

PARAMETERS:

essid : Set the ESSID (or Network Name - in some products it may also called Domain ID). The ESSID is used to identify cells which are part of the same virtual network.

Examples:

iwconfig wlan0 essid <ESSID>

rts[_threshold]: Set the RTS Threshold.

Example:

iwconfig wlan0 rts 250

frag[_threshold]: Set the Fragmentation Threshold.

Example:

iwconfig wlan0 frag 512

key/enc[ryption]:

Used to manipulate encryption or scrambling keys and encryption mode. To set the current encryption key, just enter the key in hex digits as *XXXX-XXXX-XXXX-XXXX* or *XXXXXXXX*. To set a key other than the current key, append *[index]* to the key itself. You can also enter the key as an ASCII string by using the *s:* prefix. To change which key is the current active key, just enter *[index]* (without entering any key value). *off* and *on* disable and reenable encryption, *open* set the system in open mode (accept non-encrypted packets) and *restricted* discard non-encrypted packets.

Examples :

iwconfig wlan0 key 0123-4567-89 [1]

iwconfig wlan0 key [1] open

iwconfig wlan0 key off

One should note that when using iwconfig wlan0 essid <ESSID>, we will auto join the ESSID. Because our firmware will select the right channel number, one doesn't need to set the channel by command.

3.3 Set up IP address:

If you use the RedHat distribution Linux, you can edit the /etc/sysconfig/network-scripts/ifcfg-wlan0 file to set up the IP address on booting process. Or one can use the netconfig command for ip address setting.

We provide two types setting in the following examples. One is to assign a fix IP address, netmask, and default gateway. Another is to get IP configuration from a DHCP server.

3.3.1 Fixed Setting:

This is an example of fixed IP setting

```
DEVICE='wlan0'
IPADDR='192.168.2.98'
NETMASK='255.255.255.0'
NETWORK='192.168.2.0'
```

BROADCAST='192.168.2.255'
ONBOOT='yes'
GATEWAY='192.168.2.254'

3.3.2 Get IP setting from DHCP:

This is an example of getting ip from DHCP server.

DEVICE='wlan0'
BOOTPROTO='dhcp'
ONBOOT='yes'

4. USB Compatibility:

It may get the problems that "**USB device not accepting new address= xxx** ". If this happen, one can solve this problem by substituting the usbcore modified by us. We provide usbcore of several kernel versions which are most popular seen in the Redhat distribution. One should download the usbcore package which matches your kernel. If you don't know what your kernel version is, you can type '***uname –r***'. After download the package, one just needs to do the following things.

1. **Uncompress the package:**

   tar zxvf usbcore-xxx.yyy.zzz.tar.gz

   cd usbcore

2. **Configure the system setting:**

   make config

   make all

   make install

3. **Reboot**

When doing the second step, one had better backup the original usbcore.o. The original usbcore.o is in the directory /lib/modules/$(MODULE_VERSION)/kernel/drivers/usb where $(MODULE_VERSION) is what you type '***uname –r***'.

If there is no package whose version is the same as your kernel, you can modify the usbcore.o in the kernel source tree yourself. The appendix section lists how to modify. In fact, only two files should be modified. One is the usb.c and the other is hub.c. We suggest that one copy the following files (usb.c usb-debug.c hub.c hub.h devio.c devices.c inode.c drivers.c) located in $(LINUX_SRC)/drivers/usb/ where $(LINUX_SRC) is the location of linux kernel source to a temp directory. Modify usb.c and hub.c in the temp directory. In fact, the modification in usb.c is just add a new function named usb_new_device2 and declare that it is an export function by adding this line **EXPORT_SYMBOL(usb_new_device2)**. In the hub.c, one should declare the prototype of usb_new_device2 by adding this line **int usb_new_device2(struct usb_device *);** and change the **if (!usb_new_device(dev))** statement in the function **usb_hub_port_connect_change** to

if (!usb_new_device2(dev)). We also provide a Makefile. One should only generate Makefile by the

content listed in the appendix part, and put the Makefile into the temp directory just created. Then type

1. **make**
2. **make install**

We also suggest that when doing make install, one had better backup the original usbcore.o first.

5. Conclusion:

This document doesn't explain how to setup the wireless lan environment so detailedly. One may get some problems when setting up the wireless lan environment. If you have any question about how to set up the environment, you can send an e-mail to us or find the solution on the network.

6. Reference:

[1] http://www.linux-wlan.com/ **The web site of linux-wlan.**

[2] http://www.usb.org/ **The official web site of USB**

[3] http://www.linux-usb.org/ **The resources of USB supported in Linux can be found on this web site.**

Appendix:

========== usb.c ==========

```
/*
 * By the time we get here, the device has gotten a new device ID
 * and is in the default state. We need to identify the thing and
 * get the ball rolling..
 *
 * Returns 0 for success, != 0 for error.
 */
int usb_new_device2(struct usb_device *dev)
{
        int err;
        int addr;

        /* USB v1.1 5.5.3 */
        /* We read the first 8 bytes from the device descriptor to get to */
        /*    the bMaxPacketSize0 field. Then we set the maximum packet size */
        /*    for the control pipe, and retrieve the rest */
        dev->epmaxpacketin [0] = 8;
        dev->epmaxpacketout[0] = 8;

        printk("Address: %d\n", dev->devnum);
        addr = dev->devnum;
        dev->devnum = 0;

        err = usb_get_descriptor(dev, USB_DT_DEVICE, 0, &dev->descriptor, 8);
        if (err < 8) {
                if (err < 0)
                        err("USB device not responding, giving up (error=%d)", err);
                else
                        err("USB device descriptor short read (expected %i, got %i)", 8, err);
                clear_bit(dev->devnum, &dev->bus->devmap.devicemap);
                dev->devnum = -1;
                return 1;
        }
```

```
        dev->epmaxpacketin [0] = 8;
        dev->epmaxpacketout[0] = 8;
        dev->devnum = addr;

        err = usb_set_address(dev);
        if (err < 0) {
                err("USB device not accepting new address=%d (error=%d)",
                        dev->devnum, err);
                clear_bit(dev->devnum, &dev->bus->devmap.devicemap);
                dev->devnum = -1;
                return 1;
        }

        wait_ms(10);       /* Let the SET_ADDRESS settle */

        dev->epmaxpacketin [0] = dev->descriptor.bMaxPacketSize0;
        dev->epmaxpacketout[0] = dev->descriptor.bMaxPacketSize0;

        err = usb_get_device_descriptor(dev);
        if (err < sizeof(dev->descriptor)) {
                if (err < 0)
                        err("unable to get device descriptor (error=%d)", err);
                else
                        err("USB device descriptor short read (expected %i, got %i)",
                                sizeof(dev->descriptor), err);

                clear_bit(dev->devnum, &dev->bus->devmap.devicemap);
                dev->devnum = -1;
                return 1;
        }

        wait_ms(1); /* Wait 1 ms */

        err = usb_get_configuration(dev);
        if (err < 0) {
                err("unable to get device %d configuration (error=%d)",
                        dev->devnum, err);
                clear_bit(dev->devnum, &dev->bus->devmap.devicemap);
                dev->devnum = -1;
                usb_free_dev(dev);
                return 1;
        }

        /* we set the default configuration here */
        err = usb_set_configuration(dev, dev->config[0].bConfigurationValue);
        if (err) {
                err("failed to set device %d default configuration (error=%d)",
                        dev->devnum, err);
                clear_bit(dev->devnum, &dev->bus->devmap.devicemap);
                dev->devnum = -1;
                return 1;
        }

        dbg("new device strings: Mfr=%d, Product=%d, SerialNumber=%d",
                dev->descriptor.iManufacturer, dev->descriptor.iProduct, dev->descriptor.iSerialNumber);
#ifdef DEBUG
        if (dev->descriptor.iManufacturer)
                usb_show_string(dev, "Manufacturer", dev->descriptor.iManufacturer);
        if (dev->descriptor.iProduct)
                usb_show_string(dev, "Product", dev->descriptor.iProduct);
```

```
        if (dev->descriptor.iSerialNumber)
                usb_show_string(dev, "SerialNumber", dev->descriptor.iSerialNumber);
#endif

        /* now that the basic setup is over, add a /proc/bus/usb entry */
        usbdevfs_add_device(dev);

        /* find drivers willing to handle this device */
        usb_find_drivers(dev);

        /* userspace may load modules and/or configure further */
        call_policy ("add", dev);

        return 0;
}

EXPORT_SYMBOL(usb_new_device);
EXPORT_SYMBOL(usb_new_device2)
```

========== Hub.c ==========

```
/* Declare for usb_new_device2 */

int usb_new_device2(struct usb_device *);

static void usb_hub_port_connect_change(struct usb_device *hub, int port,
                                        struct usb_port_status *portsts)
{
  …

                /* Run it through the hoops (find a driver, etc) */
-               if (!usb_new_device(dev))
+               if (!usb_new_device2(dev))
                        goto done;
  …
}
```

========== Makefile ==========

```
CC = gcc
LD = ld

MOD_VER = $(shell uname –r)
LINUX_SRC = /usr/src/linux-2.4

CFLAGS = -c -O2 -D__KERNEL__ -I$(LINUX_SRC)/include -Wall -Wstrict-prototypes\
-fomit-frame-pointer -fno-strict-aliasing -fno-common -Wno-unused -pipe\
-mpreferred-stack-boundary=2 -march=i686 -DMODULE -DMODVERSIONS\
-DEXPORT_SYMTAB -include $(LINUX_SRC)/include/linux/modversions.h

CFLAGS_EXTRA = -DCONFIG_PROC_FS

OBJECTS = $(SOURCE:%.c=%.o)
SOURCE = usb.o usb-debug.o hub.o devio.o devices.o inode.o drivers.o

usbcore: $(OBJECTS)
        $(LD) -r $(OBJECTS) -o usbcore.o

%.o: %.c
        $(CC) $(CFLAGS) $(CFLAGS_EXTRA) -o $@ $<
```

```
install:
	cp usbcore.o /lib/modules/$(MOD_VER)/kernel/drivers/usb

clean:
	rm -f *.o *~
```