

How to add wake on lan support for your card (was: Re: FreeBSD WOL sis on)

Source: <http://unix.derkeiler.com/Mailing-Lists/FreeBSD/hackers/2007-11/msg00279.html>

- *From:* Stefan Sperling <stsp@xxxxxxxxx>
 - *Date:* Sun, 25 Nov 2007 22:48:50 +0100
-

On Sun, Nov 25, 2007 at 10:53:59AM -0800, David Leslie wrote:

Linux does support WOL for (at least) this SiS900 NIC
(I can verify that it does work on this board), so
maybe it will be supported in FreeBSD in the future?

Sure, it's possible.

Adding support for a card is not that hard actually.

It can be done in one or two evenings by someone who has
messed with any kind of driver before, and maybe a week
by someone who hasn't and is willing to learn basics about
device drivers, e.g. how to access and manipulate registers from C.
Knowing C is a prerequisite of course. I myself had no idea at all
about drivers either before I started working on the WOL patch.

But I currently rarely have time to work on this (except the odd
Sunday like today), and I keep getting requests to support more
chipsets that queue up faster than I can handle :-/

Maybe if I provide a little guide some people will get interested
and help a bit.

Here it goes:

= How to add WOL support for your card =

To add WOL support for your card, and you have no data sheet
for the card, first look at the Linux driver to find out whether
the card can do WOL under Linux. There should be some routines for
configuring WOL. Look at the `ethtool_ops` struct, if the driver
sets pointers to `get_wol` and `set_wol` routines inside it, it supports WOL.

For example, the "via-rhine" driver on Linux (`if_vr` on FreeBSD)
does this:

How to add wake on lan support for your card (was: Re: FreeBSD WOL sis on)

```
static struct ethtool_ops netdev_ethtool_ops = {  
// snip  
.get_wol = rhine_get_wol,  
.set_wol = rhine_set_wol,  
// snip
```

Now that you know that you can get this to work, apply the WOL patch to your system: <http://stsp.næme/wol/>
See the README.txt for instructions.

== How do I tell the chip on my card to do WOL? ==

You can do this in a new driver routine called `dev_enable_wol()`, where `dev_` is the usual driver name prefix, e.g. `xl_enable_wol()` for `if_xl`.

If you have a data sheet, it will probably tell you how to enable WOL in detail. The data sheet for my NatSemi card is freely available on the web and has a very good section on configuring wake on lan:

<http://www.national.com/ds/DP/DP83815.pdf>

Look at page 90 in that PDF. Even if your chip is different the general procedure is likely the same on most chips so this serves as a good example (along with a working implementation for the `if_sis` driver in my patch). That page helped me getting started at lot.

How exactly WOL is configured depends on the device, but usually involves

- a) configuring the receive filter
- b) some WOL configuration register
- c) putting the device into D3 ("sleep") power state

=== The receive filter ==

The receive filter will simply need to accept *any* kind of packet.

There is usually a register in the chip with bits that define what kind packets the receiver on the card should hand on to the driver and which it should just ignore — typically the filter understands unicast, multicast and broadcast packet types.

In case of the 3com chip, the Linux driver (3c59x) does this in the `acpi_set_WOL()` function to set the bits in the receive register:

```
/* The RxFilter must accept the WOL frames. */  
iowrite16(SetRxFilter|RxStation|RxMulticast|RxBroadcast, iaddr + EL3_CMD);  
iowrite16(RxEnable, iaddr + EL3_CMD);
```

How to add wake on lan support for your card (was: Re: FreeBSD WOL sis on)

The FreeBSD (almost) equivalent in `xl_enable_wol()` is this:

```
/* Configure the receive filter to accept any kind of packet. */
XL_SEL_WIN(5);
rxfilt = CSR_READ_1(sc, XL_W5_RX_FILTER);
rxfilt |= XL_RXFILTER_INDIVIDUAL | XL_RXFILTER_ALLMULTI |
XL_RXFILTER_BROADCAST | XL_RXFILTER_ALLFRAMES;
CSR_WRITE_2(sc, XL_COMMAND, XL_CMD_RX_SET_FILT | rxfilt);
```

Of course, register business is highly chip specific. For example, other chips I've dealt with didn't have register windows at all so the `XL_SEL_WIN` isn't needed there, and the bit mask macros for the receive filter config register will of course be named somewhat differently in each driver.

Of course, a data sheet helps. If you don't have one, looking at what the Linux driver does and matching register offsets and bit masks with the FreeBSD driver will probably help.

=== The WOL config register ===

The WOL config register is usually a 16bit (e.g. `if_xl`) or 32bit (e.g. `if_sis`) register somewhere in the chip memory with each bit corresponding to a certain type of wake event — magic packet, link status change, unicast packet reception, broadcast packet reception, etc.

The device will only wake the system if an event happens that it has been configured for.

Using the NatSemi card an example, there is a 32bit WOL configuration register at offset `0x40` in the chip register space and bit number 9 (the 8th bit from the right) tells the chip to wake up if it receives a magic packet. See the DP83815 data sheet again, page 55.

So to add wake on magic packet support, we add the following to `/usr/src/sys/dev/if_sisreg.h`:

```
/* NS DP83815/6 registers */
// other registers omitted...
#define NS_WCSR 0x40

/* Bits in DP83815 Wake On Lan Command/Status register */
#define NS_WCSR_WAKE_MAGIC 0x00000200
```

Of course there are cards that can do more than just magic packets, and no two users want to wake up their boxes the same way, and some

How to add wake on lan support for your card (was: Re: FreeBSD WOL sis on)

don't want WOL at all, so ifconfig has been extended to allow WOL events to be configured (default is to not wake the box at all).

Each driver adds a small field to its softc where it remembers what WOL events the user wants configured. For example, the xl_softc (defined in /usr/src/sys/pci/if_xlreg.h on RELENG_6, and in /usr/src/sys/pci/if_xl.c on RELENG_7) got the following additional field:

```
struct xl_softc {
// lots of other fields omitted...
uint32_t xl_wol_events;
};
```

Also, we define a macro shorthand that defines what kind of wake events our driver can tell the card to look out for, for example:

```
#define XL_SUPPORTED_WOL_EVENTS (IFWOL_WAKE_ON_MAGIC | IFWOL_WAKE_ON_LINK)
```

The wake event types known to the system are listed in /usr/include/net/if.h:

```
IFWOL_WAKE_ON_UNICAST
IFWOL_WAKE_ON_MULTICAST
IFWOL_WAKE_ON_BROADCAST
IFWOL_WAKE_ON_MAGIC
IFWOL_WAKE_ON_LINK
```

(Note that I plan to remove the IFWOL_ENABLE_SOPASSWD flag, so while it is still listed in if.h currently, I omitted it above.)

The ifconfig<->driver communication is done via a set of new ioctls, defined in /usr/include/sys/sockio.h:

```
#define SIOCGIFWOL_OPTS _IOWR('i', 125, struct ifreq) /* get wake on lan
options */
#define SIOCSIFWOL_OPTS _IOW('i', 126, struct ifreq) /* set wake on lan
options */
#define SIOCGIFWOL_SUPP _IOWR('i', 127, struct ifreq) /* get wake on lan
modes supported by
device */
```

So whenever the user runs one of
ifconfig xl0
ifconfig xl0 wakeon magic
or similar your driver's ioctl handler will be called and you will get one of the three requests above.

ioctl arguments are usually evaluated in a big switch statement. The following was added to the if_xl driver's xl_ioctl() routine so that the driver remembers what events the user wants to be

How to add wake on lan support for your card (was: Re: FreeBSD WOL sis on)

configured in the chip (SIOCGIFWOLOPTS and SIOCSIFWOLOPTS),
^Get ^Set
and allowing ifconfig to determine what type of events the device
supports (SIOCGIFWOLSUPP):

```
case SIOCGIFWOLSUPP:
if (sc->xl_flags & XL_FLAG_SUPPORTS_WOL)
ifr->ifr_wolopts.ifwol_supported =
XL_SUPPORTED_WOL_EVENTS;
else
ifr->ifr_wolopts.ifwol_supported = 0;
error = 0;
break;
case SIOCGIFWOLOPTS:
XL_LOCK(sc);
xl_get_wolopts(sc, &ifr->ifr_wolopts);
XL_UNLOCK(sc);
error = 0;
break;
case SIOCSIFWOLOPTS:
XL_LOCK(sc);
error = xl_set_wolopts(sc, &ifr->ifr_wolopts);
XL_UNLOCK(sc);
break;
```

The xl_(get|set)_wolopts() routines do some sanity checking
and store/return the value of ifr_wolopts in the xl_wol_events
field in the softc. [The naming is horribly inconsistent here,
I plan to clean this up in the future so everything is called
simply "wolcfg" everywhere instead of "wolopts" at some places
and "wol_events" at others.]

After dealing with the receive filter, xl_enable_wol() checks what
the user wants configured and writes the corresponding values into
the WOL config register (in this case using the CSR_WRITE_2 macro
to access registers the 16bit WOL config register on the chip):

```
/* Configure wake on lan events. */
config = 0;
XL_SEL_WIN(7);
if (sc->xl_wol_events & IFWOL_WAKE_ON_MAGIC)
config |= XL_WAKE_ON_MAGIC;
if (sc->xl_wol_events & IFWOL_WAKE_ON_LINK)
config |= XL_WAKE_ON_LINK;
CSR_WRITE_2(sc, XL_W7_BM_WOL, config);
```

=== Putting the device into D3 power state ===

This is not strictly device specific, at least for PCI devices
there is an API that drivers can (should?) use to set the

How to add wake on lan support for your card (was: Re: FreeBSD WOL sis on)

device into D3 power state, see `pci_set_powerstate(9)`

Try that and see if it works.

Currently the `if_xl` driver uses this instead, which should have the same effect:

```
/* Make sure power management is enabled and set the card
 * into D3hot power state, so it stays active after system shutdown. */
config = pci_read_config(dev, XL_PCI_PWRMGMTCTRL, 2);
config |= XL_PME_EN | XL_PSTATE_D3;
pci_write_config(dev, XL_PCI_PWRMGMTCTRL, config | XL_PME_EN, 2);
```

However, doing it manually as done in `if_sis` works fine, too.
(Except I guess there may probably be races if the bus is currently busy or so, maybe some PCI expert can shed some light on whether the following is good practice or not...)

```
/* Set appropriate power state, so the card stays active
 * after system shutdown. */
CSR_WRITE_4(sc, NS_CLKRUN, NS_CLKRUN_PMESTS | NS_CLKRUN_PMEENB);
```

=== Where to call `dev_enable_wol()` ===

Every driver has a shutdown routine. Add your `enable_wol()` routine to the very end of it so that it is the very last routine that runs in the driver at system shutdown time. For example, the `xl_shutdown()` routine looks like this:

```
static void
xl_shutdown(device_t dev)
{
    struct xl_softc *sc;

    sc = device_get_softc(dev);

    XL_LOCK(sc);
    xl_reset(sc);
    xl_stop(sc);
    xl_enable_wol(dev);
    XL_UNLOCK(sc);
}
```

Make sure that you do not call the routine outside of a `LOCK()` and `UNLOCK()` scope, otherwise people with more than one CPU in their computer might complain.

=== That was it ===

How to add wake on lan support for your card (was: Re: FreeBSD WOL sis on)

Well, I hope I haven't missed anything important, but I guess that's about it.

This should get every aspiring kernel hacker started. For more details look at the patch and see what it does for the drivers it already supports. You should be able to understand the patch fairly well now. Especially the `sis_enable_wol()` routine in `if_sis` should be relatively clear quickly with the help of the data sheet.

If you're already a hardcore kernel hacker why are you still reading this guide and not hacking up a patch already? :)

I will happily accept diffs against drivers.

If you have questions just ask.

Good luck,

--

stefan

<http://stsp.name> PGP Key: 0xF59D25F0

Attachment: pgpcKYvDQoDhL.pgp

Description: PGP signature