

Re: [patch 2.6.20-rc3 1/3] rtc-cmos driver

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2007-01/msg01508.html>

- *From:* David Brownell <david-b@xxxxxxxxxxxx>
 - *Date:* Fri, 5 Jan 2007 19:10:01 -0800
-

On Friday 05 January 2007 12:45 pm, Alessandro Zummo wrote:

On Fri, 5 Jan 2007 10:01:57 -0800
David Brownell <david-b@xxxxxxxxxxxx> wrote:

This is an "RTC framework" driver for the "CMOS" RTCs which are standard on PCs and some other platforms. That's MC146818 compatible silicon. Advantages of this vs. drivers/char/rtc.c (use one or the other, only one will be able to claim the RTC irq) include:

Hi David,

good code and well commented, thank you.

Glad to hear it!

I only have some comments:

– I would put anything that is x86 related (pnp,acpi) in a separate file so that people working on non x86 systems can have a better grasp of the driver.

I believe that splitting this driver into multiple files would create more confusion than it solves. Bus glue is actually a very small part of the code, and it's cleanly split out. So I'm not keen on this change at all.

And having done a grep of the source tree, splitting out PNP bus glue (and platform bus glue?) into a separate file isn't common. IDE does, and 8250; both cases with a *very* complex core, shared with many drivers. But the normal case #ifdefs PNP support, combining it with alternative bus glue which is all too often nasty legacy "probe the hardware" logic.

Re: [patch 2.6.20-rc3 1/3] rtc-cmos driver

(Also: there are no longer any ACPI calls in this driver; and do recall that at least ia64 uses ACPI too. So `_none_` of that code is x86-specific...)

– the name should be `rtc-mc146818` to be coherent with the other drivers, but this can cause confusion.

Yes, I used the "rtc-cmos" name to minimize confusion. It's a generic name; I've even seen multiple books, including some southbridge docs, talking about "CMOS RTC" or "CMOS clock".

"MC146818" is easy to mistype, hard to say, obsolete (!), and virtually unused in most technical contexts. But "cmos clock" is widely understood, at least in PC-derived contexts, and make sense talking about most any southbridge.

I guess I'm surprised you called `rtc-m48t86` that, instead of using the name `rtc-mc146818` (since ST markets that M48 chip as a replacement for the `mc146818`)... :)

I'm not deeply attached to this name, but I couldn't come up with a better generic name.

– please put yourself in `MODULE_AUTHOR`

OK. Appended.

other than that, I'm fine with the code.

I'd appreciate if someone (Woody?) can test this code on ARM.

There are PPC, M68K, SPARC, and other boards that could also use this; ARMs tend to integrate some other RTC on-chip. But on whatever non-PC platform is involved in such sanity testing, that involves adding a `platform_device` to board setup code.

– Dave

===== CUT HERE

This is an "RTC framework" driver for the "CMOS" RTCs which are standard

Re: [patch 2.6.20-rc3 1/3] rtc-cmos driver

Re: [patch 2.6.20-rc3 1/3] rtc-cmos driver

on PCs and some other platforms. That's MC146818 compatible silicon. Advantages of this vs. drivers/char/rtc.c (use one _or_ the other, only one will be able to claim the RTC irq) include:

- This leverages both the new RTC framework and the driver model; both PNPACPI and platform device modes are supported. (A separate patch creates a platform device on PCs where PNPACPI isn't configured.)
- It supports common extensions like longer alarms. (A separate patch exports that information from ACPI through platform_data.)
- Likewise, system wakeup events use "real driver model support", with policy control via sysfs "wakeup" attributes and using normal rtc ioctls to manage wakeup. (Patch in the works. The ACPI hooks are known; /proc/acpi/alarm can vanish. Making it work with EFI will be a minor challenge to someone with e.g. a MiniMac.)

It's not yet been tested on non-x86 systems, without ACPI, or with HPET. And the RTC framework will surely have teething pains on "mainstream" PC-based systems (though most embedded Linux systems use it heavily), not limited to sorting out the "/dev/rtc0" issue (udev easily tweaked). Also, the ALSA rtctimer code doesn't use the new RTC API.

Otherwise, this should be a no-known-regressions replacement for the old drivers/char/rtc.c driver, and should help the non-embedded distros (and the new timekeeping code) start to switch to the framework.

Signed-off-by: David Brownell <dbrownell@xxxxxxxxxxxxxxxxxxxxxxxx>

Note also that any systems using "rtc-m48t86" are candidates to switch over to this more functional driver; the platform data is different, and the way bytes are read is different, but otherwise those chips should be compatible.

```

drivers/rtc/Kconfig | 13
drivers/rtc/Makefile | 1
drivers/rtc/rtc-cmos.c | 718 +++++
include/linux/mc146818rtc.h | 10
4 files changed, 742 insertions(+)

```

Index: g26/drivers/rtc/Kconfig

```

=====
--- g26.orig/drivers/rtc/Kconfig 2007-01-02 19:24:32.000000000 -0800
+++ g26/drivers/rtc/Kconfig 2007-01-02 23:35:41.000000000 -0800
@@ -95,6 +95,19 @@ config RTC_INTF_DEV_UIE_EMUL
comment "RTC drivers"
depends on RTC_CLASS

+config RTC_DRV_CMOS
+ tristate "CMOS real time clock"
+ depends on RTC_CLASS && (X86_PC || ACPI)

```

Re: [patch 2.6.20-rc3 1/3] rtc-cmos driver

- + help
- + Say "yes" here to get direct support for the real time clock
- + found in every PC or ACPI-based system, and some others.
- + Specifically the original MC146818, compatibles like those
- + in PC south bridges, the DS12887 or M48T86, some LPC bus
- + chips, and so on.
- + This driver can also be built as a module. If so, the module
- + will be called rtc-cmos.

+
config RTC_DRV_X1205
tristate "Xicor/Intersil X1205"
depends on RTC_CLASS && I2C
Index: g26/drivers/rtc/Makefile

```
=====
--- g26.orig/drivers/rtc/Makefile 2007-01-02 19:24:29.000000000 -0800
+++ g26/drivers/rtc/Makefile 2007-01-02 23:35:41.000000000 -0800
@@ -15,6 +15,7 @@ obj-$(CONFIG_RTC_INTF_SYSFS) += rtc-sysf
obj-$(CONFIG_RTC_INTF_PROC) += rtc-proc.o
obj-$(CONFIG_RTC_INTF_DEV) += rtc-dev.o
```

```
+obj-$(CONFIG_RTC_DRV_CMOS) += rtc-cmos.o
obj-$(CONFIG_RTC_DRV_X1205) += rtc-x1205.o
obj-$(CONFIG_RTC_DRV_ISL1208) += rtc-isl1208.o
obj-$(CONFIG_RTC_DRV_TEST) += rtc-test.o
Index: g26/include/linux/mc146818rtc.h
```

```
=====
--- g26.orig/include/linux/mc146818rtc.h 2007-01-02 19:24:29.000000000 -0800
+++ g26/include/linux/mc146818rtc.h 2007-01-02 23:35:41.000000000 -0800
@@ -18,6 +18,16 @@
#ifdef __KERNEL__
#include <linux/spinlock.h> /* spinlock_t */
extern spinlock_t rtc_lock; /* serialize CMOS RAM access */
+
+/* Some RTCs extend the mc146818 register set to support alarms of more
+ * than 24 hours in the future; or dates that include a century code.
+ * This platform_data structure can pass this information to the driver.
+ */
+struct cmos_rtc_board_info {
+ u8 rtc_day_alarm; /* zero, or register index */
+ u8 rtc_mon_alarm; /* zero, or register index */
+ u8 rtc_century; /* zero, or register index */
+};
#endif
```

```
/******
Index: g26/drivers/rtc/rtc-cmos.c
```

```
=====
--- /dev/null 1970-01-01 00:00:00.000000000 +0000
+++ g26/drivers/rtc/rtc-cmos.c 2007-01-05 18:17:21.000000000 -0800
@@ -0,0 +1,722 @@
```

```

+/*
+ * RTC class driver for "CMOS RTC": PCs, ACPI, etc
+ *
+ * Copyright (C) 1996 Paul Gortmaker (drivers/char/rtc.c)
+ * Copyright (C) 2006 David Brownell (convert to new framework)
+ *
+ * This program is free software; you can redistribute it and/or
+ * modify it under the terms of the GNU General Public License
+ * as published by the Free Software Foundation; either version
+ * 2 of the License, or (at your option) any later version.
+ */
+
+/*
+ * The original "cmos clock" chip was an MC146818 chip, now obsolete.
+ * That defined the register interface now provided by all PCs, some
+ * non-PC systems, and incorporated into ACPI. Modern PC chipsets
+ * integrate an MC146818 clone in their southbridge, and boards use
+ * that instead of discrete clones like the DS12887 or M48T86. There
+ * are also clones that connect using the LPC bus.
+ *
+ * That register API is also used directly by various other drivers
+ * (notably for integrated NVRAM), infrastructure (x86 has code to
+ * bypass the RTC framework, directly reading the RTC during boot
+ * and updating minutes/seconds for systems using NTP synch) and
+ * utilities (like userspace 'hwclock', if no /dev node exists).
+ *
+ * So ALL calls to CMOS_READ and CMOS_WRITE must be done with
+ * interrupts disabled, holding the global rtc_lock, to exclude those
+ * other drivers and utilities on correctly configured systems.
+ */
+#include <linux/kernel.h>
+#include <linux/module.h>
+#include <linux/init.h>
+#include <linux/interrupt.h>
+#include <linux/spinlock.h>
+#include <linux/platform_device.h>
+#include <linux/mod_devicetable.h>
+
+#include <asm/rtc.h>
+
+struct cmos_rtc {
+ struct rtc_device *rtc;
+ struct device *dev;
+ int irq;
+ struct resource *iomem;
+
+ u8 suspend_ctrl;
+
+ /* newer hardware extends the original register set */
+ u8 day_alarm;

```

```

+ u8 mon_alm;
+ u8 century;
+};
+
+/* both platform and pnp busses use negative numbers for invalid irqs */
+#define is_valid_irq(n) ((n) >= 0)
+
+static const char driver_name[] = "rtc_cmos";
+
+/*-----*/
+
+static int cmos_read_time(struct device *dev, struct rtc_time *t)
+{
+ /* REVISIT: if the clock has a "century" register, use
+ * that instead of the heuristic in get_rtc_time().
+ * That'll make Y3K compatibility (year > 2070) easy!
+ */
+ get_rtc_time(t);
+ return 0;
+}
+
+static int cmos_set_time(struct device *dev, struct rtc_time *t)
+{
+ /* REVISIT: set the "century" register if available
+ *
+ * NOTE: this ignores the issue whereby updating the seconds
+ * takes effect exactly 500ms after we write the register.
+ * (Also queuing and other delays before we get this far.)
+ */
+ return set_rtc_time(t);
+}
+
+static int cmos_read_alarm(struct device *dev, struct rtc_wkalrm *t)
+{
+ struct cmos_rtc *cmos = dev_get_drvdata(dev);
+ unsigned char rtc_control;
+
+ if (!is_valid_irq(cmos->irq))
+ return -EIO;
+
+ /* Basic alarms only support hour, minute, and seconds fields.
+ * Some also support day and month, for alarms up to a year in
+ * the future.
+ */
+ t->time.tm_mday = -1;
+ t->time.tm_mon = -1;
+
+ spin_lock_irq(&rtc_lock);
+ t->time.tm_sec = CMOS_READ(RTC_SECONDS_ALARM);
+ t->time.tm_min = CMOS_READ(RTC_MINUTES_ALARM);
+ t->time.tm_hour = CMOS_READ(RTC_HOURS_ALARM);

```

```

+
+ if (cmos->day_alm) {
+ t->time.tm_mday = CMOS_READ(cmos->day_alm);
+ if (!t->time.tm_mday)
+ t->time.tm_mday = -1;
+
+ if (cmos->mon_alm) {
+ t->time.tm_mon = CMOS_READ(cmos->mon_alm);
+ if (!t->time.tm_mon)
+ t->time.tm_mon = -1;
+ }
+ }
+
+ rtc_control = CMOS_READ(RTC_CONTROL);
+ spin_unlock_irq(&rtc_lock);
+
+ /* REVISIT this assumes PC style usage: always BCD */
+
+ if (((unsigned)t->time.tm_sec) < 0x60)
+ t->time.tm_sec = BCD2BIN(t->time.tm_sec);
+ else
+ t->time.tm_sec = -1;
+ if (((unsigned)t->time.tm_min) < 0x60)
+ t->time.tm_min = BCD2BIN(t->time.tm_min);
+ else
+ t->time.tm_min = -1;
+ if (((unsigned)t->time.tm_hour) < 0x24)
+ t->time.tm_hour = BCD2BIN(t->time.tm_hour);
+ else
+ t->time.tm_hour = -1;
+
+ if (cmos->day_alm) {
+ if (((unsigned)t->time.tm_mday) <= 0x31)
+ t->time.tm_mday = BCD2BIN(t->time.tm_mday);
+ else
+ t->time.tm_mday = -1;
+ if (cmos->mon_alm) {
+ if (((unsigned)t->time.tm_mon) <= 0x12)
+ t->time.tm_mon = BCD2BIN(t->time.tm_mon) - 1;
+ else
+ t->time.tm_mon = -1;
+ }
+ }
+ t->time.tm_year = -1;
+
+ t->enabled = !(rtc_control & RTC_AIE);
+ t->pending = 0;
+
+ return 0;
+ }
+

```

Re: [patch 2.6.20-rc3 1/3] rtc-cmos driver

```
+static int cmos_set_alarm(struct device *dev, struct rtc_wkalrm *t)
+{
+ struct cmos_rtc *cmos = dev_get_drvdata(dev);
+ unsigned char mon, mday, hrs, min, sec;
+ unsigned char rtc_control, rtc_intr;
+
+ if (!is_valid_irq(cmos->irq))
+ return -EIO;
+
+ /* REVISIT this assumes PC style usage: always BCD */
+
+ /* Writing 0xff means "don't care" or "match all". */
+
+ mon = t->time.tm_mon;
+ mon = (mon < 12) ? BIN2BCD(mon) : 0xff;
+ mon++;
+
+ mday = t->time.tm_mday;
+ mday = (mday >= 1 && mday <= 31) ? BIN2BCD(mday) : 0xff;
+
+ hrs = t->time.tm_hour;
+ hrs = (hrs < 24) ? BIN2BCD(hrs) : 0xff;
+
+ min = t->time.tm_min;
+ min = (min < 60) ? BIN2BCD(min) : 0xff;
+
+ sec = t->time.tm_sec;
+ sec = (sec < 60) ? BIN2BCD(sec) : 0xff;
+
+ spin_lock_irq(&rtc_lock);
+
+ /* next rtc irq must not be from previous alarm setting */
+ rtc_control = CMOS_READ(RTC_CONTROL);
+ rtc_control &= ~RTC_AIE;
+ CMOS_WRITE(rtc_control, RTC_CONTROL);
+ rtc_intr = CMOS_READ(RTC_INTR_FLAGS);
+ if (rtc_intr)
+ rtc_update_irq(&cmos->rtc->class_dev, 1, rtc_intr);
+
+ /* update alarm */
+ CMOS_WRITE(hrs, RTC_HOURS_ALARM);
+ CMOS_WRITE(min, RTC_MINUTES_ALARM);
+ CMOS_WRITE(sec, RTC_SECONDS_ALARM);
+
+ /* the system may support an "enhanced" alarm */
+ if (cmos->day_alm) {
+ CMOS_WRITE(mday, cmos->day_alm);
+ if (cmos->mon_alm)
+ CMOS_WRITE(mon, cmos->mon_alm);
+ }
+
+ }
```

```

+ if (t->enabled) {
+ rtc_control |= RTC_AIE;
+ CMOS_WRITE(rtc_control, RTC_CONTROL);
+ rtc_intr = CMOS_READ(RTC_INTR_FLAGS);
+ if (rtc_intr)
+ rtc_update_irq(&cmos->rtc->class_dev, 1, rtc_intr);
+ }
+
+ spin_unlock_irq(&rtc_lock);
+
+ return 0;
+}
+
+static int cmos_set_freq(struct device *dev, int freq)
+{
+ struct cmos_rtc *cmos = dev_get_drvdata(dev);
+ int f;
+ unsigned long flags;
+
+ if (!is_valid_irq(cmos->irq))
+ return -ENXIO;
+
+ /* 0 = no irqs; 1 = 2^15 Hz ... 15 = 2^0 Hz */
+ f = ffs(freq);
+ if (f != 0) {
+ if (f-- > 16 || freq != (1 << f))
+ return -EINVAL;
+ f = 16 - f;
+ }
+
+ spin_lock_irqsave(&rtc_lock, flags);
+ CMOS_WRITE(RTC_REF_CLK_32KHZ | f, RTC_FREQ_SELECT);
+ spin_unlock_irqrestore(&rtc_lock, flags);
+
+ return 0;
+}
+
+#if defined(CONFIG_RTC_INTF_DEV) || defined(CONFIG_RTC_INTF_DEV_MODULE)
+
+static int
+cmos_rtc_ioctl(struct device *dev, unsigned int cmd, unsigned long arg)
+{
+ struct cmos_rtc *cmos = dev_get_drvdata(dev);
+ unsigned char rtc_control, rtc_intr;
+ unsigned long flags;
+
+ switch (cmd) {
+ case RTC_AIE_OFF:
+ case RTC_AIE_ON:
+ case RTC_UIE_OFF:
+ case RTC_UIE_ON:

```

```

+ case RTC_PIE_OFF:
+ case RTC_PIE_ON:
+ if (!is_valid_irq(cmos->irq))
+ return -EINVAL;
+ break;
+ default:
+ return -ENOIOCTLCMD;
+ }
+
+ spin_lock_irqsave(&rtc_lock, flags);
+ rtc_control = CMOS_READ(RTC_CONTROL);
+ switch (cmd) {
+ case RTC_AIE_OFF: /* alarm off */
+ rtc_control &= ~RTC_AIE;
+ break;
+ case RTC_AIE_ON: /* alarm on */
+ rtc_control |= RTC_AIE;
+ break;
+ case RTC_UIE_OFF: /* update off */
+ rtc_control &= ~RTC_UIE;
+ break;
+ case RTC_UIE_ON: /* update on */
+ rtc_control |= RTC_UIE;
+ break;
+ case RTC_PIE_OFF: /* periodic off */
+ rtc_control &= ~RTC_PIE;
+ break;
+ case RTC_PIE_ON: /* periodic on */
+ rtc_control |= RTC_PIE;
+ break;
+ }
+ CMOS_WRITE(rtc_control, RTC_CONTROL);
+ rtc_intr = CMOS_READ(RTC_INTR_FLAGS);
+ if (rtc_intr)
+ rtc_update_irq(&cmos->rtc->class_dev, 1, rtc_intr);
+ spin_unlock_irqrestore(&rtc_lock, flags);
+ return 0;
+}
+
+ #else
+ #define cmos_rtc_ioctl NULL
+ #endif
+
+ #if defined(CONFIG_RTC_INTF_PROC) || defined(CONFIG_RTC_INTF_PROC_MODULE)
+
+ static int cmos_procfs(struct device *dev, struct seq_file *seq)
+ {
+ struct cmos_rtc *cmos = dev_get_drvdata(dev);
+ unsigned char rtc_control, valid;
+
+ spin_lock_irq(&rtc_lock);

```

```

+ rtc_control = CMOS_READ(RTC_CONTROL);
+ valid = CMOS_READ(RTC_VALID);
+ spin_unlock_irq(&rtc_lock);
+
+ /* NOTE: at least ICH6 reports battery status using a different
+ * (non-RTC) bit; and SQWE is ignored on many current systems.
+ */
+ return seq_printf(seq,
+ "periodic_IRQ\t: %s\n"
+ "update_IRQ\t: %s\n"
+ // "square_wave\t: %s\n"
+ // "BCD\t\t: %s\n"
+ "DST_enable\t: %s\n"
+ "periodic_freq\t: %d\n"
+ "batt_status\t: %s\n",
+ (rtc_control & RTC_PIE) ? "yes" : "no",
+ (rtc_control & RTC_UIE) ? "yes" : "no",
+ // (rtc_control & RTC_SQWE) ? "yes" : "no",
+ // (rtc_control & RTC_DM_BINARY) ? "no" : "yes",
+ (rtc_control & RTC_DST_EN) ? "yes" : "no",
+ cmos->rtc->irq_freq,
+ (valid & RTC_VRT) ? "okay" : "dead");
+ }
+
+ #else
+ #define cmos_procfs NULL
+ #endif
+
+ static const struct rtc_class_ops cmos_rtc_ops = {
+ .ioctl = cmos_rtc_ioctl,
+ .read_time = cmos_read_time,
+ .set_time = cmos_set_time,
+ .read_alarm = cmos_read_alarm,
+ .set_alarm = cmos_set_alarm,
+ .proc = cmos_procfs,
+ .irq_set_freq = cmos_set_freq,
+ };
+
+ /*-----*/
+
+ static struct cmos_rtc cmos_rtc;
+
+ static irqreturn_t cmos_interrupt(int irq, void *p)
+ {
+ u8 irqstat;
+
+ spin_lock(&rtc_lock);
+ irqstat = CMOS_READ(RTC_INTR_FLAGS);
+ spin_unlock(&rtc_lock);
+
+ if (irqstat) {

```

```

+ /* NOTE: irqstat may have e.g. RTC_PF set
+ * even when RTC_PIE is clear...
+ */
+ rtc_update_irq(p, 1, irqstat);
+ return IRQ_HANDLED;
+ } else
+ return IRQ_NONE;
+ }
+
+ #ifdef CONFIG_PNPACPI
+ #define is_pnpacpi() 1
+ #define INITSECTION
+
+ #else
+ #define is_pnpacpi() 0
+ #define INITSECTION __init
+ #endif
+
+ static int INITSECTION
+ cmos_do_probe(struct device *dev, struct resource *ports, int rtc_irq)
+ {
+ struct cmos_rtc_board_info *info = dev->platform_data;
+ int retval = 0;
+ unsigned char rtc_control;
+
+ /* there can be only one ... */
+ if (cmos_rtc.dev)
+ return -EBUSY;
+
+ if (!ports)
+ return -ENODEV;
+
+ cmos_rtc.irq = rtc_irq;
+ cmos_rtc.iomem = ports;
+
+ if (info) {
+ cmos_rtc.day_alarm = info->rtc_day_alarm;
+ cmos_rtc.mon_alarm = info->rtc_mon_alarm;
+ cmos_rtc.century = info->rtc_century;
+ }
+
+ cmos_rtc.rtc = rtc_device_register(driver_name, dev,
+ &cmos_rtc_ops, THIS_MODULE);
+ if (IS_ERR(cmos_rtc.rtc))
+ return PTR_ERR(cmos_rtc.rtc);
+
+ cmos_rtc.dev = dev;
+ dev_set_drvdata(dev, &cmos_rtc);
+
+ /* platform and pnp busses handle resources incompatibly.
+ */

```

```

+ * REVISIT for non-x86 systems we may need to handle io memory
+ * resources: ioremap them, and request_mem_region().
+ */
+ if (is_pnpacpi()) {
+   retval = request_resource(&ioport_resource, ports);
+   if (retval < 0) {
+     dev_dbg(dev, "i/o registers already in use\n");
+     goto cleanup0;
+   }
+ }
+ rename_region(ports, cmos_rtc.rtc->class_dev.class_id);
+
+ spin_lock_irq(&rtc_lock);
+
+ /* force periodic irq to CMOS reset default of 1024Hz;
+ *
+ * REVISIT it's been reported that at least one x86_64 ALI mobo
+ * doesn't use 32KHz here ... for portability we might need to
+ * do something about other clock frequencies.
+ */
+ CMOS_WRITE(RTC_REF_CLK_32KHZ | 0x06, RTC_FREQ_SELECT);
+ cmos_rtc.rtc->irq_freq = 1024;
+
+ /* disable irqs.
+ *
+ * NOTE after changing RTC_xIE bits we always read INTR_FLAGS;
+ * allegedly some older rtc's need that to handle irqs properly
+ */
+ rtc_control = CMOS_READ(RTC_CONTROL);
+ rtc_control &= ~(RTC_PIE | RTC_AIE | RTC_UIE);
+ CMOS_WRITE(rtc_control, RTC_CONTROL);
+ CMOS_READ(RTC_INTR_FLAGS);
+
+ spin_unlock_irq(&rtc_lock);
+
+ /* FIXME teach the alarm code how to handle binary mode;
+ * <asm-generic/rtc.h> doesn't know 12-hour mode either.
+ */
+ if (!(rtc_control & RTC_24H) || (rtc_control & (RTC_DM_BINARY))) {
+   dev_dbg(dev, "only 24-hr BCD mode supported\n");
+   retval = -ENXIO;
+   goto cleanup1;
+ }
+
+ if (is_valid_irq(rtc_irq))
+   retval = request_irq(rtc_irq, cmos_interrupt, IRQF_DISABLED,
+   cmos_rtc.rtc->class_dev.class_id,
+   &cmos_rtc.rtc->class_dev);
+   if (retval < 0) {
+     dev_dbg(dev, "IRQ %d is already in use\n", rtc_irq);
+     goto cleanup1;
+   }

```

```

+ }
+
+ /* REVISIT optionally make 50 or 114 bytes NVRAM available,
+ * like rtc-ds1553, rtc-ds1742 ... this will often include
+ * registers for century, and day/month alarm.
+ */
+
+ pr_info("%s: alarms up to one %s%\n",
+ cmos_rtc.rtc->class_dev.class_id,
+ is_valid_irq(rtc_irq)
+ ? (cmos_rtc.mon_alm
+ ? "year"
+ : (cmos_rtc.day_alm
+ ? "month" : "day"))
+ : "no",
+ cmos_rtc.century ? ", y3k" : "");
+ );
+
+ return 0;
+
+ cleanup1:
+ rename_region(ports, NULL);
+ cleanup0:
+ rtc_device_unregister(cmos_rtc.rtc);
+ return retval;
+ }
+
+ static void cmos_do_shutdown(void)
+ {
+ unsigned char rtc_control;
+
+ spin_lock_irq(&rtc_lock);
+ rtc_control = CMOS_READ(RTC_CONTROL);
+ rtc_control &= ~(RTC_PIE|RTC_AIE|RTC_UIE);
+ CMOS_WRITE(rtc_control, RTC_CONTROL);
+ CMOS_READ(RTC_INTR_FLAGS);
+ spin_unlock_irq(&rtc_lock);
+ }
+
+ static void __exit cmos_do_remove(struct device *dev)
+ {
+ struct cmos_rtc *cmos = dev_get_drvdata(dev);
+
+ cmos_do_shutdown();
+
+ if (is_pnpacpi())
+ release_resource(cmos->iomem);
+ rename_region(cmos->iomem, NULL);
+
+ if (is_valid_irq(cmos->irq))
+ free_irq(cmos->irq, &cmos_rtc.rtc->class_dev);

```

```

+
+ rtc_device_unregister(cmos_rtc.rtc);
+
+ cmos_rtc.dev = NULL;
+ dev_set_drvdata(dev, NULL);
+}
+
+ #ifdef CONFIG_PM
+
+ static int cmos_suspend(struct device *dev, pm_message_t mesg)
+ {
+     struct cmos_rtc *cmos = dev_get_drvdata(dev);
+     int do_wake = device_may_wakeup(dev);
+     unsigned char tmp, irqstat;
+
+     /* only the alarm might be a wakeup event source */
+     spin_lock_irq(&rtc_lock);
+     cmos->suspend_ctrl = tmp = CMOS_READ(RTC_CONTROL);
+     if (tmp & (RTC_PIE|RTC_AIE|RTC_UIE)) {
+         if (do_wake)
+             tmp &= ~(RTC_PIE|RTC_UIE);
+         else
+             tmp &= ~(RTC_PIE|RTC_AIE|RTC_UIE);
+         CMOS_WRITE(tmp, RTC_CONTROL);
+         irqstat = CMOS_READ(RTC_INTR_FLAGS);
+     } else
+         irqstat = 0;
+     spin_unlock_irq(&rtc_lock);
+
+     if (irqstat)
+         rtc_update_irq(&cmos->rtc->class_dev, 1, irqstat);
+
+     /* ACPI HOOK: enable ACPI_EVENT_RTC when (tmp & RTC_AIE)
+      * ... it'd be best if we could do that under rtc_lock.
+      */
+
+     pr_debug("%s: suspend%s, ctrl %02x\n",
+             cmos_rtc.rtc->class_dev.class_id,
+             (tmp & RTC_AIE) ? ", alarm may wake" : "",
+             tmp);
+
+     return 0;
+ }
+
+ static int cmos_resume(struct device *dev)
+ {
+     struct cmos_rtc *cmos = dev_get_drvdata(dev);
+     unsigned char tmp = cmos->suspend_ctrl;
+
+     /* REVISIT: a mechanism to resync the system clock (jiffies)
+      * on resume should be portable between platforms ...

```

```

+ */
+
+ /* re-enable any irqs previously active */
+ if (tmp & (RTC_PIE|RTC_AIE|RTC_UIE)) {
+
+ /* ACPI HOOK: disable ACPI_EVENT_RTC when (tmp & RTC_AIE) */
+
+ spin_lock_irq(&rtc_lock);
+ CMOS_WRITE(tmp, RTC_CONTROL);
+ tmp = CMOS_READ(RTC_INTR_FLAGS);
+ spin_unlock_irq(&rtc_lock);
+ if (tmp)
+ rtc_update_irq(&cmos->rtc->class_dev, 1, tmp);
+ }
+
+ pr_debug("%s: resume, ctrl %02x\n",
+ cmos_rtc.rtc->class_dev.class_id,
+ cmos->suspend_ctrl);
+
+
+ return 0;
+}
+
+ #else
+ #define cmos_suspend NULL
+ #define cmos_resume NULL
+ #endif
+
+ /*-----*/
+
+ /* On ACPI systems, the device node may be created as either a PNP
+ * device or a platform_device. In either case the FADT data should
+ * have been transferred to us through platform_data.
+ */
+
+ #ifdef CONFIG_PNPACPI
+ #include <linux/pnp.h>
+
+ static int __devinit
+ cmos_pnp_probe(struct pnp_dev *pnp, const struct pnp_device_id *id)
+ {
+ /* REVISIT paranoia argues for a shutdown notifier, since PNP
+ * drivers can't provide shutdown() methods to disable IRQs.
+ * Or better yet, fix PNP to allow those methods...
+ */
+ return cmos_do_probe(&pnp->dev,
+ &pnp->res.port_resource[0],
+ pnp->res.irq_resource[0].start);
+ }
+

```

```

+static void __exit cmos_pnp_remove(struct pnp_dev *pnp)
+{
+ cmos_do_remove(&pnp->dev);
+}
+
+#ifdef CONFIG_PM
+
+static int cmos_pnp_suspend(struct pnp_dev *pnp, pm_message_t mesg)
+{
+ return cmos_suspend(&pnp->dev, mesg);
+}
+
+static int cmos_pnp_resume(struct pnp_dev *pnp)
+{
+ return cmos_resume(&pnp->dev);
+}
+
+#else
+#define cmos_pnp_suspend NULL
+#define cmos_pnp_resume NULL
+#endif
+
+
+static const struct pnp_device_id rtc_ids[] = {
+ { .id = "PNP0b00", },
+ { .id = "PNP0b01", },
+ { .id = "PNP0b02", },
+ { },
+};
+MODULE_DEVICE_TABLE(pnp, rtc_ids);
+
+static struct pnp_driver cmos_pnp_driver = {
+ .name = (char *) driver_name,
+ .id_table = rtc_ids,
+ .probe = cmos_pnp_probe,
+ .remove = __exit_p(cmos_pnp_remove),
+
+ /* flag ensures resume() gets called, and stops syslog spam */
+ .flags = PNP_DRIVER_RES_DO_NOT_CHANGE,
+ .suspend = cmos_pnp_suspend,
+ .resume = cmos_pnp_resume,
+};
+
+static int __init cmos_init(void)
+{
+ return pnp_register_driver(&cmos_pnp_driver);
+}
+module_init(cmos_init);
+
+static void __exit cmos_exit(void)
+{

```

```

+ pnp_unregister_driver(&cmos_pnp_driver);
+}
+module_exit(cmos_exit);
+
+/* no PNPACPI */
+
+/*-----*/
+
+/* Platform setup should have set up an RTC device, when PNPACPI is
+ * unavailable ... including non-PC and non-ACPI platforms.
+ */
+
+static int __init cmos_platform_probe(struct platform_device *pdev)
+{
+ return cmos_do_probe(&pdev->dev,
+ platform_get_resource(pdev, IORESOURCE_IO, 0),
+ platform_get_irq(pdev, 0));
+}
+
+static int __exit cmos_platform_remove(struct platform_device *pdev)
+{
+ cmos_do_remove(&pdev->dev);
+ return 0;
+}
+
+static void cmos_platform_shutdown(struct platform_device *pdev)
+{
+ cmos_do_shutdown();
+}
+
+static struct platform_driver cmos_platform_driver = {
+ .remove = __exit_p(cmos_platform_remove),
+ .shutdown = cmos_platform_shutdown,
+ .driver = {
+ .name = (char *) driver_name,
+ .suspend = cmos_suspend,
+ .resume = cmos_resume,
+ }
+};
+
+static int __init cmos_init(void)
+{
+ return platform_driver_probe(&cmos_platform_driver,
+ cmos_platform_probe);
+}
+module_init(cmos_init);
+
+static void __exit cmos_exit(void)
+{
+ platform_driver_unregister(&cmos_platform_driver);
+}

```

Re: [patch 2.6.20-rc3 1/3] rtc-cmos driver

```
+module_exit(cmos_exit);  
+  
+  
+#endif /* !PNPACPI */  
+  
+MODULE_AUTHOR("David Brownell");  
+MODULE_DESCRIPTION("CMOS' RTC driver: PCs, ACPI, etc");  
+MODULE_LICENSE("GPL");  
-
```

To unsubscribe from this list: send the line "unsubscribe linux-kernel" in the body of a message to majordomo@xxxxxxxxxxxxxxxxxxx
More majordomo info at <http://vger.kernel.org/majordomo-info.html>
Please read the FAQ at <http://www.tux.org/lkml/>