

[patch 2.6.27-rc8-git] add drivers/mfd/twl4030-core.c

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2008-10/msg02034.html>

- *From:* David Brownell <david-b@xxxxxxxxxxxx>
- *Date:* Mon, 6 Oct 2008 10:47:49 -0700

From: David Brownell <dbrownell@xxxxxxxxxxxxxxxxxxxxxxxx>

This patch adds the core of the TWL4030 driver, which supports chips including the TPS65950. These chips are multi-function; see

<http://focus.ti.com/docs/prod/folders/print/tps65950.html>

Public specs are in the works. For now, the block diagram on the second page of the datasheet is fairly informative.

There are some known issues with this core code. Most notably, the IRQ dispatching needs simplification (to use more of genirq), generalization (integrating support for secondary IRQ dispatch as well as primary, and removing the build dependency on OMAP), and then probably updating to leverage threaded IRQ support (expected to arrive in mainline "soon").

Once the core is in mainline, drivers for other parts of this chip can follow its lead and start swimming upstream too.

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

```

---
drivers/mfd/Kconfig | 14
drivers/mfd/Makefile | 2
drivers/mfd/twl4030-core.c | 1257 +++++
include/linux/i2c/twl4030.h | 339 +++++
4 files changed, 1612 insertions(+)

```

```

--- a/drivers/mfd/Kconfig
+++ b/drivers/mfd/Kconfig
@@ -50,6 +50,20 @@ config HTC_PASIC3
HTC Magician devices, respectively. Actual functionality is
handled by the leds-pasic3 and ds1wm drivers.

```

```

+config TWL4030_CORE
+bool "Texas Instruments TWL4030/TPS659x0 Support"
+depends on I2C=y && GENERIC_HARDIRQS && (ARCH_OMAP2 || ARCH_OMAP3)
+help

```

[patch 2.6.27-rc8-git] add drivers/mfd/twl4030-core.c

```
+ Say yes here if you have TWL4030 family chip on your board.
+ This core driver provides register access and IRQ handling
+ facilities, and registers devices for the various functions
+ so that function-specific drivers can bind to them.
+
+ These multi-function chips are found on many OMAP2 and OMAP3
+ boards, providing power management, RTC, GPIO, keypad, a
+ high speed USB OTG transceiver, an audio codec (on most
+ versions) and many other features.
+
config MFD_TMIO
bool
default n
--- a/drivers/mfd/Makefile
+++ b/drivers/mfd/Makefile
@@ -12,6 +12,8 @@ obj-$(CONFIG_MFD_T7L66XB) += t7l66xb.o
obj-$(CONFIG_MFD_TC6387XB) += tc6387xb.o
obj-$(CONFIG_MFD_TC6393XB) += tc6393xb.o

+obj-$(CONFIG_TWL4030_CORE) += twl4030-core.o
+
obj-$(CONFIG_MFD_CORE) += mfd-core.o

obj-$(CONFIG_MCP) += mcp-core.o
--- /dev/null
+++ b/drivers/mfd/twl4030-core.c
@@ -0,0 +1,1257 @@
+/*
+ * twl4030_core.c - driver for TWL4030/TPS659x0 PM and audio CODEC devices
+ *
+ * Copyright (C) 2005-2006 Texas Instruments, Inc.
+ *
+ * Modifications to defer interrupt handling to a kernel thread:
+ * Copyright (C) 2006 MontaVista Software, Inc.
+ *
+ * Based on tlv320aic23.c:
+ * Copyright (c) by Kai Svahn <kai.svahn@xxxxxxxx>
+ *
+ * Code cleanup and modifications to IRQ handler.
+ * by syed khasim <x0khasim@xxxxxx>
+ *
+ * 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.
+ *
+ * This program is distributed in the hope that it will be useful,
+ * but WITHOUT ANY WARRANTY; without even the implied warranty of
+ * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
+ * GNU General Public License for more details.
+ *
+ */
```

[patch 2.6.27-rc8-git] add drivers/mfd/twl4030-core.c

```
+ * You should have received a copy of the GNU General Public License
+ * along with this program; if not, write to the Free Software
+ * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
+ */
+
+#include <linux/kernel_stat.h>
+#include <linux/init.h>
+#include <linux/mutex.h>
+#include <linux/interrupt.h>
+#include <linux/irq.h>
+#include <linux/random.h>
+#include <linux/kthread.h>
+#include <linux/platform_device.h>
+#include <linux/clk.h>
+
+#include <linux/i2c.h>
+#include <linux/i2c/twl4030.h>
+
+/*
+ * The TWL4030 "Triton 2" is one of a family of a multi-function "Power
+ * Management and System Companion Device" chips originally designed for
+ * use in OMAP2 and OMAP 3 based systems. Its control interfaces use I2C,
+ * often at around 3 Mbit/sec, including for interrupt handling.
+ *
+ * This driver core provides genirq support for the interrupts emitted,
+ * by the various modules, and exports register access primitives.
+ *
+ * FIXME this driver currently requires use of the first interrupt line
+ * (and associated registers).
+ */
+#define DRIVER_NAME "twl4030"
+
+#if defined(CONFIG_TWL4030_BCI_BATTERY) || \
+ defined(CONFIG_TWL4030_BCI_BATTERY_MODULE)
+#define twl_has_bci() true
+#else
+#define twl_has_bci() false
+#endif
+
+#if defined(CONFIG_KEYBOARD_TWL4030) || defined(CONFIG_KEYBOARD_TWL4030_MODULE)
+#define twl_has_keypad() true
+#else
+#define twl_has_keypad() false
+#endif
+
+#if defined(CONFIG_GPIO_TWL4030) || defined(CONFIG_GPIO_TWL4030_MODULE)
+#define twl_has_gpio() true
+#else
+#define twl_has_gpio() false
```

```
+#endif
+
+#if defined(CONFIG_TWL4030_MADC) || defined(CONFIG_TWL4030_MADC_MODULE)
+#define twl_has_madc() true
+#else
+#define twl_has_madc() false
+#endif
+
+#if defined(CONFIG_RTC_DRV_TWL4030) || defined(CONFIG_RTC_DRV_TWL4030_MODULE)
+#define twl_has_rtc() true
+#else
+#define twl_has_rtc() false
+#endif
+
+#if defined(CONFIG_TWL4030_USB) || defined(CONFIG_TWL4030_USB_MODULE)
+#define twl_has_usb() true
+#else
+#define twl_has_usb() false
+#endif
+
+static inline void activate_irq(int irq)
+{
+#ifdef CONFIG_ARM
+ /* ARM requires an extra step to clear IRQ_NOREQUEST, which it
+ * sets on behalf of every irq_chip. Also sets IRQ_NOPROBE.
+ */
+ set_irq_flags(irq, IRQF_VALID);
+#else
+ /* same effect on other architectures */
+ set_irq_noprobe(irq);
+#endif
+}
+
+/* Primary Interrupt Handler on TWL4030 Registers */
+
+/* Register Definitions */
+
+#define REG_PIH_ISR_P1 (0x1)
+#define REG_PIH_ISR_P2 (0x2)
+#define REG_PIH_SIR (0x3)
+
+/* Triton Core internal information (BEGIN) */
+
+/* Last - for index max*/
+#define TWL4030_MODULE_LAST TWL4030_MODULE_SECURED_REG
+
+#define TWL4030_NUM_SLAVES 4
+
+/* Base Address defns for twl4030_map[] */
+
```

[patch 2.6.27-rc8-git] add drivers/mfd/twl4030-core.c

```
+/* subchip/slave 0 – USB ID */
+#define TWL4030_BASEADD_USB 0x0000
+
+/* subchip/slave 1 – AUD ID */
+#define TWL4030_BASEADD_AUDIO_VOICE 0x0000
+#define TWL4030_BASEADD_GPIO 0x0098
+#define TWL4030_BASEADD_INTBR 0x0085
+#define TWL4030_BASEADD_PIH 0x0080
+#define TWL4030_BASEADD_TEST 0x004C
+
+/* subchip/slave 2 – AUX ID */
+#define TWL4030_BASEADD_INTERRUPTS 0x00B9
+#define TWL4030_BASEADD_LED 0x00EE
+#define TWL4030_BASEADD_MADC 0x0000
+#define TWL4030_BASEADD_MAIN_CHARGE 0x0074
+#define TWL4030_BASEADD_PRECHARGE 0x00AA
+#define TWL4030_BASEADD_PWM0 0x00F8
+#define TWL4030_BASEADD_PWM1 0x00FB
+#define TWL4030_BASEADD_PWMA 0x00EF
+#define TWL4030_BASEADD_PWMB 0x00F1
+#define TWL4030_BASEADD_KEYPAD 0x00D2
+
+/* subchip/slave 3 – POWER ID */
+#define TWL4030_BASEADD_BACKUP 0x0014
+#define TWL4030_BASEADD_INT 0x002E
+#define TWL4030_BASEADD_PM_MASTER 0x0036
+#define TWL4030_BASEADD_PM_RECEIVER 0x005B
+#define TWL4030_BASEADD_RTC 0x001C
+#define TWL4030_BASEADD_SECURED_REG 0x0000
+
+/* Triton Core internal information (END) */
+
+
+/* Few power values */
+#define R_CFG_BOOT 0x05
+#define R_PROTECT_KEY 0x0E
+
+/* access control values for R_PROTECT_KEY */
+#define KEY_UNLOCK1 0xce
+#define KEY_UNLOCK2 0xec
+#define KEY_LOCK 0x00
+
+
+/* some fields in R_CFG_BOOT */
+#define HFCLK_FREQ_19p2_MHZ (1 << 0)
+#define HFCLK_FREQ_26_MHZ (2 << 0)
+#define HFCLK_FREQ_38p4_MHZ (3 << 0)
+#define HIGH_PERF_SQ (1 << 3)
+
+
+/*-----*/
+
```

[patch 2.6.27-rc8-git] add drivers/mfd/twl4030-core.c

```
+/**
+ * struct twl4030_mod_iregs - TWL module IMR/ISR regs to mask/clear at init
+ * @mod_no: TWL4030 module number (e.g., TWL4030_MODULE_GPIO)
+ * @sih_ctrl: address of module SIH_CTRL register
+ * @reg_cnt: number of IMR/ISR regs
+ * @imrs: pointer to array of TWL module interrupt mask register indices
+ * @isrs: pointer to array of TWL module interrupt status register indices
+ *
+ * Ties together TWL4030 modules and lists of IMR/ISR registers to mask/clear
+ * during twl_init_irq().
+ */
+struct twl4030_mod_iregs {
+ const u8 mod_no;
+ const u8 sih_ctrl;
+ const u8 reg_cnt;
+ const u8 *imrs;
+ const u8 *isrs;
+};
+
+/* TWL4030 INT module interrupt mask registers */
+static const u8 __initconst twl4030_int_imr_regs[] = {
+ TWL4030_INT_PWR_IMR1,
+ TWL4030_INT_PWR_IMR2,
+};
+
+/* TWL4030 INT module interrupt status registers */
+static const u8 __initconst twl4030_int_isr_regs[] = {
+ TWL4030_INT_PWR_ISR1,
+ TWL4030_INT_PWR_ISR2,
+};
+
+/* TWL4030 INTERRUPTS module interrupt mask registers */
+static const u8 __initconst twl4030_interrupts_imr_regs[] = {
+ TWL4030_INTERRUPTS_BCIIMR1A,
+ TWL4030_INTERRUPTS_BCIIMR1B,
+ TWL4030_INTERRUPTS_BCIIMR2A,
+ TWL4030_INTERRUPTS_BCIIMR2B,
+};
+
+/* TWL4030 INTERRUPTS module interrupt status registers */
+static const u8 __initconst twl4030_interrupts_isr_regs[] = {
+ TWL4030_INTERRUPTS_BCIISR1A,
+ TWL4030_INTERRUPTS_BCIISR1B,
+ TWL4030_INTERRUPTS_BCIISR2A,
+ TWL4030_INTERRUPTS_BCIISR2B,
+};
+
+/* TWL4030 MADC module interrupt mask registers */
+static const u8 __initconst twl4030_madc_imr_regs[] = {
+ TWL4030_MADC_IMR1,
+ TWL4030_MADC_IMR2,
```

```
+};
+
+/* TWL4030 MADC module interrupt status registers */
+static const u8 __initconst twl4030_madc_isr_regs[] = {
+ TWL4030_MADC_ISR1,
+ TWL4030_MADC_ISR2,
+};
+
+/* TWL4030 keypad module interrupt mask registers */
+static const u8 __initconst twl4030_keypad_imr_regs[] = {
+ TWL4030_KEYPAD_KEYP_IMR1,
+ TWL4030_KEYPAD_KEYP_IMR2,
+};
+
+/* TWL4030 keypad module interrupt status registers */
+static const u8 __initconst twl4030_keypad_isr_regs[] = {
+ TWL4030_KEYPAD_KEYP_ISR1,
+ TWL4030_KEYPAD_KEYP_ISR2,
+};
+
+/* TWL4030 GPIO module interrupt mask registers */
+static const u8 __initconst twl4030_gpio_imr_regs[] = {
+ REG_GPIO_IMR1A,
+ REG_GPIO_IMR1B,
+ REG_GPIO_IMR2A,
+ REG_GPIO_IMR2B,
+ REG_GPIO_IMR3A,
+ REG_GPIO_IMR3B,
+};
+
+/* TWL4030 GPIO module interrupt status registers */
+static const u8 __initconst twl4030_gpio_isr_regs[] = {
+ REG_GPIO_ISR1A,
+ REG_GPIO_ISR1B,
+ REG_GPIO_ISR2A,
+ REG_GPIO_ISR2B,
+ REG_GPIO_ISR3A,
+ REG_GPIO_ISR3B,
+};
+
+/* TWL4030 modules that have IMR/ISR registers that must be masked/cleared */
+static const struct twl4030_mod_iregs __initconst twl4030_mod_regs[] = {
+ {
+ .mod_no = TWL4030_MODULE_INT,
+ .sih_ctrl = TWL4030_INT_PWR_SIH_CTRL,
+ .reg_cnt = ARRAY_SIZE(twl4030_int_imr_regs),
+ .imrs = twl4030_int_imr_regs,
+ .isrs = twl4030_int_isr_regs,
+ },
+ {
+ .mod_no = TWL4030_MODULE_INTERRUPTS,
```

```

+ .sih_ctrl = TWL4030_INTERRUPTS_BCISIHCTRL,
+ .reg_cnt = ARRAY_SIZE(twl4030_interrupts_imr_regs),
+ .imrs = twl4030_interrupts_imr_regs,
+ .isrs = twl4030_interrupts_isr_regs,
+ },
+ {
+ .mod_no = TWL4030_MODULE_MADC,
+ .sih_ctrl = TWL4030_MADC_SIH_CTRL,
+ .reg_cnt = ARRAY_SIZE(twl4030_madc_imr_regs),
+ .imrs = twl4030_madc_imr_regs,
+ .isrs = twl4030_madc_isr_regs,
+ },
+ {
+ .mod_no = TWL4030_MODULE_KEYPAD,
+ .sih_ctrl = TWL4030_KEYPAD_KEYP_SIH_CTRL,
+ .reg_cnt = ARRAY_SIZE(twl4030_keypad_imr_regs),
+ .imrs = twl4030_keypad_imr_regs,
+ .isrs = twl4030_keypad_isr_regs,
+ },
+ {
+ .mod_no = TWL4030_MODULE_GPIO,
+ .sih_ctrl = REG_GPIO_SIH_CTRL,
+ .reg_cnt = ARRAY_SIZE(twl4030_gpio_imr_regs),
+ .imrs = twl4030_gpio_imr_regs,
+ .isrs = twl4030_gpio_isr_regs,
+ },
+ };
+
+/*-----*/
+
+/* is driver active, bound to a chip? */
+static bool inuse;
+
+/* Structure for each TWL4030 Slave */
+struct twl4030_client {
+ struct i2c_client *client;
+ u8 address;
+
+ /* max numb of i2c_msg required is for read =2 */
+ struct i2c_msg xfer_msg[2];
+
+ /* To lock access to xfer_msg */
+ struct mutex xfer_lock;
+ };
+
+static struct twl4030_client twl4030_modules[TWL4030_NUM_SLAVES];
+
+
+/* mapping the module id to slave id and base address */
+struct twl4030mapping {
+ unsigned char sid; /* Slave ID */

```

```

+ unsigned char base; /* base address */
+ };
+
+static struct twl4030mapping twl4030_map[TWL4030_MODULE_LAST + 1] = {
+ /*
+ * NOTE: don't change this table without updating the
+ * <linux/i2c/twl4030.h> defines for TWL4030_MODULE_*
+ * so they continue to match the order in this table.
+ */
+
+ { 0, TWL4030_BASEADD_USB },
+
+ { 1, TWL4030_BASEADD_AUDIO_VOICE },
+ { 1, TWL4030_BASEADD_GPIO },
+ { 1, TWL4030_BASEADD_INTBR },
+ { 1, TWL4030_BASEADD_PIH },
+ { 1, TWL4030_BASEADD_TEST },
+
+ { 2, TWL4030_BASEADD_KEYPAD },
+ { 2, TWL4030_BASEADD_MADC },
+ { 2, TWL4030_BASEADD_INTERRUPTS },
+ { 2, TWL4030_BASEADD_LED },
+ { 2, TWL4030_BASEADD_MAIN_CHARGE },
+ { 2, TWL4030_BASEADD_PRECHARGE },
+ { 2, TWL4030_BASEADD_PWM0 },
+ { 2, TWL4030_BASEADD_PWM1 },
+ { 2, TWL4030_BASEADD_PWMA },
+ { 2, TWL4030_BASEADD_PWMB },
+
+ { 3, TWL4030_BASEADD_BACKUP },
+ { 3, TWL4030_BASEADD_INT },
+ { 3, TWL4030_BASEADD_PM_MASTER },
+ { 3, TWL4030_BASEADD_PM_RECEIVER },
+ { 3, TWL4030_BASEADD_RTC },
+ { 3, TWL4030_BASEADD_SECURED_REG },
+ };
+
+ /*-----*/
+
+ /*
+ * TWL4030 doesn't have PIH mask, hence dummy function for mask
+ * and unmask of the (eight) interrupts reported at that level ...
+ * masking is only available from SIH (secondary) modules.
+ */
+
+static void twl4030_i2c_ackirq(unsigned int irq)
+{
+}
+
+static void twl4030_i2c_disableint(unsigned int irq)
+{

```

```

+}
+
+static void twl4030_i2c_enableint(unsigned int irq)
+{
+}
+
+static struct irq_chip twl4030_irq_chip = {
+ .name = "twl4030",
+ .ack = twl4030_i2c_ackirq,
+ .mask = twl4030_i2c_disableint,
+ .unmask = twl4030_i2c_enableint,
+};
+
+/*-----*/
+
+/* Exported Functions */
+
+/**
+ * twl4030_i2c_write - Writes a n bit register in TWL4030
+ * @mod_no: module number
+ * @value: an array of num_bytes+1 containing data to write
+ * @reg: register address (just offset will do)
+ * @num_bytes: number of bytes to transfer
+ *
+ * IMPORTANT: for 'value' parameter: Allocate value num_bytes+1 and
+ * valid data starts at Offset 1.
+ *
+ * Returns the result of operation - 0 is success
+ */
+int twl4030_i2c_write(u8 mod_no, u8 *value, u8 reg, u8 num_bytes)
+{
+ int ret;
+ int sid;
+ struct twl4030_client *twl;
+ struct i2c_msg *msg;
+
+ if (unlikely(mod_no > TWL4030_MODULE_LAST)) {
+ pr_err("%s: invalid module number %d\n", DRIVER_NAME, mod_no);
+ return -EPERM;
+ }
+ sid = twl4030_map[mod_no].sid;
+ twl = &twl4030_modules[sid];
+
+ if (unlikely(!inuse)) {
+ pr_err("%s: client %d is not initialized\n", DRIVER_NAME, sid);
+ return -EPERM;
+ }
+ mutex_lock(&twl->xfer_lock);
+ /*
+ * [MSG1]: fill the register address data
+ * fill the data Tx buffer

```

```

+ */
+ msg = &twl->xfer_msg[0];
+ msg->addr = twl->address;
+ msg->len = num_bytes + 1;
+ msg->flags = 0;
+ msg->buf = value;
+ /* over write the first byte of buffer with the register address */
+ *value = twl4030_map[mod_no].base + reg;
+ ret = i2c_transfer(twl->client->adapter, twl->xfer_msg, 1);
+ mutex_unlock(&twl->xfer_lock);
+
+ /* i2cTransfer returns num messages.translate it pls.. */
+ if (ret >= 0)
+ ret = 0;
+ return ret;
+ }
+EXPORT_SYMBOL(twl4030_i2c_write);
+
+/**
+ * twl4030_i2c_read - Reads a n bit register in TWL4030
+ * @mod_no: module number
+ * @value: an array of num_bytes containing data to be read
+ * @reg: register address (just offset will do)
+ * @num_bytes: number of bytes to transfer
+ *
+ * Returns result of operation - num_bytes is success else failure.
+ */
+int twl4030_i2c_read(u8 mod_no, u8 *value, u8 reg, u8 num_bytes)
+{
+ int ret;
+ u8 val;
+ int sid;
+ struct twl4030_client *twl;
+ struct i2c_msg *msg;
+
+ if (unlikely(mod_no > TWL4030_MODULE_LAST)) {
+ pr_err("%s: invalid module number %d\n", DRIVER_NAME, mod_no);
+ return -EPERM;
+ }
+ sid = twl4030_map[mod_no].sid;
+ twl = &twl4030_modules[sid];
+
+ if (unlikely(!inuse)) {
+ pr_err("%s: client %d is not initialized\n", DRIVER_NAME, sid);
+ return -EPERM;
+ }
+ mutex_lock(&twl->xfer_lock);
+ /* [MSG1] fill the register address data */
+ msg = &twl->xfer_msg[0];
+ msg->addr = twl->address;
+ msg->len = 1;

```

```

+ msg->flags = 0; /* Read the register value */
+ val = twl4030_map[mod_no].base + reg;
+ msg->buf = &val;
+ /* [MSG2] fill the data rx buffer */
+ msg = &twl->xfer_msg[1];
+ msg->addr = twl->address;
+ msg->flags = I2C_M_RD; /* Read the register value */
+ msg->len = num_bytes; /* only n bytes */
+ msg->buf = value;
+ ret = i2c_transfer(twl->client->adapter, twl->xfer_msg, 2);
+ mutex_unlock(&twl->xfer_lock);
+
+ /* i2cTransfer returns num messages.translate it pls.. */
+ if (ret >= 0)
+ ret = 0;
+ return ret;
+}
+EXPORT_SYMBOL(twl4030_i2c_read);
+
+/**
+ * twl4030_i2c_write_u8 - Writes a 8 bit register in TWL4030
+ * @mod_no: module number
+ * @value: the value to be written 8 bit
+ * @reg: register address (just offset will do)
+ *
+ * Returns result of operation - 0 is success
+ */
+int twl4030_i2c_write_u8(u8 mod_no, u8 value, u8 reg)
+{
+
+ /* 2 bytes offset 1 contains the data offset 0 is used by i2c_write */
+ u8 temp_buffer[2] = { 0 };
+ /* offset 1 contains the data */
+ temp_buffer[1] = value;
+ return twl4030_i2c_write(mod_no, temp_buffer, reg, 1);
+}
+EXPORT_SYMBOL(twl4030_i2c_write_u8);
+
+/**
+ * twl4030_i2c_read_u8 - Reads a 8 bit register from TWL4030
+ * @mod_no: module number
+ * @value: the value read 8 bit
+ * @reg: register address (just offset will do)
+ *
+ * Returns result of operation - 0 is success
+ */
+int twl4030_i2c_read_u8(u8 mod_no, u8 *value, u8 reg)
+{
+ return twl4030_i2c_read(mod_no, value, reg, 1);
+}
+EXPORT_SYMBOL(twl4030_i2c_read_u8);

```

```

+
+/*-----*/
+
+/*
+ * do_twl4030_module_irq() is the desc->handle method for each of the twl4030
+ * module interrupts that doesn't chain to another irq_chip (GPIO, power, etc).
+ * It executes in kernel thread context. On entry, cpu interrupts are disabled.
+ */
+static void do_twl4030_module_irq(unsigned int irq, irq_desc_t *desc)
+{
+ struct irqaction *action;
+ const unsigned int cpu = smp_processor_id();
+
+ /*
+ * Earlier this was desc->triggered = 1;
+ */
+ desc->status |= IRQ_LEVEL;
+
+ /*
+ * The desc->handle method would normally call the desc->chip->ack
+ * method here, but we won't bother since our ack method is NULL.
+ */
+
+ if (!desc->depth) {
+ kstat_cpu(cpu).irqs[irq]++;
+
+ action = desc->action;
+ if (action) {
+ int ret;
+ int status = 0;
+ int retval = 0;
+
+ local_irq_enable();
+
+ do {
+ /* Call the ISR with cpu interrupts enabled */
+ ret = action->handler(irq, action->dev_id);
+ if (ret == IRQ_HANDLED)
+ status |= action->flags;
+ retval |= ret;
+ action = action->next;
+ } while (action);
+
+ if (status & IRQF_SAMPLE_RANDOM)
+ add_interrupt_randomness(irq);
+
+ local_irq_disable();
+
+ if (retval != IRQ_HANDLED)
+ printk(KERN_ERR "ISR for TWL4030 module"
+ " irq %d can't handle interrupt\n",

```

```
+ irq);
+
+ /*
+ * Here is where we should call the unmask method, but
+ * again we won't bother since it is NULL.
+ */
+ } else
+ printk(KERN_CRIT "TWL4030 module irq %d has no ISR"
+ " but can't be masked!\n", irq);
+ } else
+ printk(KERN_CRIT "TWL4030 module irq %d is disabled but can't"
+ " be masked!\n", irq);
+ }
+
+static unsigned twl4030_irq_base;
+
+static struct completion irq_event;
+
+/*
+ * This thread processes interrupts reported by the Primary Interrupt Handler.
+ */
+static int twl4030_irq_thread(void *data)
+{
+ long irq = (long)data;
+ irq_desc_t *desc = irq_desc + irq;
+ static unsigned i2c_errors;
+ const static unsigned max_i2c_errors = 100;
+
+ daemonize("twl4030-irq");
+ current->flags |= PF_NOFREEZE;
+
+ while (!kthread_should_stop()) {
+ int ret;
+ int module_irq;
+ u8 pih_isr;
+
+ /* Wait for IRQ, then read PIH irq status (also blocking) */
+ wait_for_completion_interruptible(&irq_event);
+
+ ret = twl4030_i2c_read_u8(TWL4030_MODULE_PIH, &pih_isr,
+ REG_PIH_ISR_P1);
+ if (ret) {
+ pr_warning("%s: I2C error %d reading PIH ISR\n",
+ DRIVER_NAME, ret);
+ if (++i2c_errors >= max_i2c_errors) {
+ printk(KERN_ERR "Maximum I2C error count"
+ " exceeded. Terminating %s.\n",
+ __func__);
+ break;
+ }
+ }
+ complete(&irq_event);

```

```

+ continue;
+ }
+
+ /* these handlers deal with the relevant SIH irq status */
+ local_irq_disable();
+ for (module_irq = twl4030_irq_base;
+      pih_isr;
+      pih_isr >>= 1, module_irq++) {
+ if (pih_isr & 0x1) {
+ irq_desc_t *d = irq_desc + module_irq;
+
+ d->handle_irq(module_irq, d);
+ }
+ }
+ local_irq_enable();
+
+ desc->chip->unmask(irq);
+ }
+
+ return 0;
+ }
+
+ /*
+ * do_twl4030_irq() is the desc->handle method for the twl4030 interrupt.
+ * This is a chained interrupt, so there is no desc->action method for it.
+ * Now we need to query the interrupt controller in the twl4030 to determine
+ * which module is generating the interrupt request. However, we can't do i2c
+ * transactions in interrupt context, so we must defer that work to a kernel
+ * thread. All we do here is acknowledge and mask the interrupt and wakeup
+ * the kernel thread.
+ */
+static void do_twl4030_irq(unsigned int irq, irq_desc_t *desc)
+{
+ const unsigned int cpu = smp_processor_id();
+
+ /*
+ * Earlier this was desc->triggered = 1;
+ */
+ desc->status |= IRQ_LEVEL;
+
+ /*
+ * Acknowledge, clear _AND_ disable the interrupt.
+ */
+ desc->chip->ack(irq);
+
+ if (!desc->depth) {
+ kstat_cpu(cpu).irqs[irq]++;
+
+ complete(&irq_event);
+ }
+ }

```

```

+
+static struct task_struct * __init start_twl4030_irq_thread(long irq)
+{
+ struct task_struct *thread;
+
+ init_completion(&irq_event);
+ thread = kthread_run(twl4030_irq_thread, (void *)irq,
+ "twl4030 irq %ld", irq);
+ if (!thread)
+ pr_err("%s: could not create twl4030 irq %ld thread!\n",
+ DRIVER_NAME, irq);
+
+ return thread;
+}
+
+/*-----*/
+
+static int add_children(struct twl4030_platform_data *pdata)
+{
+ struct platform_device *pdev = NULL;
+ struct twl4030_client *twl = NULL;
+ int status = 0;
+
+ if (twl_has_bci() && pdata->bci) {
+ twl = &twl4030_modules[3];
+
+ pdev = platform_device_alloc("twl4030_bci", -1);
+ if (!pdev) {
+ pr_debug("%s: can't alloc bci dev\n", DRIVER_NAME);
+ status = -ENOMEM;
+ goto err;
+ }
+
+ if (status == 0) {
+ pdev->dev.parent = &twl->client->dev;
+ status = platform_device_add_data(pdev, pdata->bci,
+ sizeof(*pdata->bci));
+ if (status < 0) {
+ dev_dbg(&twl->client->dev,
+ "can't add bci data, %d\n",
+ status);
+ goto err;
+ }
+ }
+
+ if (status == 0) {
+ struct resource r = {
+ .start = TWL4030_PWRIRQ_CHG_PRE,
+ .flags = IORESOURCE_IRQ,
+ };
+
+

```

```
+ status = platform_device_add_resources(pdev, &r, 1);
+ }
+
+ if (status == 0)
+ status = platform_device_add(pdev);
+
+ if (status < 0) {
+ platform_device_put(pdev);
+ dev_dbg(&twl->client->dev,
+ "can't create bci dev, %d\n",
+ status);
+ goto err;
+ }
+ }
+
+ if (twl_has_gpio() && pdata->gpio) {
+ twl = &twl4030_modules[1];
+
+ pdev = platform_device_alloc("twl4030_gpio", -1);
+ if (!pdev) {
+ pr_debug("%s: can't alloc gpio dev\n", DRIVER_NAME);
+ status = -ENOMEM;
+ goto err;
+ }
+
+ /* more driver model init */
+ if (status == 0) {
+ pdev->dev.parent = &twl->client->dev;
+ /* device_init_wakeup(&pdev->dev, 1); */
+
+ status = platform_device_add_data(pdev, pdata->gpio,
+ sizeof(*pdata->gpio));
+ if (status < 0) {
+ dev_dbg(&twl->client->dev,
+ "can't add gpio data, %d\n",
+ status);
+ goto err;
+ }
+ }
+
+ /* GPIO module IRQ */
+ if (status == 0) {
+ struct resource r = {
+ .start = pdata->irq_base + 0,
+ .flags = IORESOURCE_IRQ,
+ };
+
+ status = platform_device_add_resources(pdev, &r, 1);
+ }
+
+ if (status == 0)
```

```
+ status = platform_device_add(pdev);
+
+ if (status < 0) {
+ platform_device_put(pdev);
+ dev_dbg(&twl->client->dev,
+ "can't create gpio dev, %d\n",
+ status);
+ goto err;
+ }
+ }
+
+ if (twl_has_keypad() && pdata->keypad) {
+ pdev = platform_device_alloc("twl4030_keypad", -1);
+ if (pdev) {
+ twl = &twl4030_modules[2];
+ pdev->dev.parent = &twl->client->dev;
+ device_init_wakeup(&pdev->dev, 1);
+ status = platform_device_add_data(pdev, pdata->keypad,
+ sizeof(*pdata->keypad));
+ if (status < 0) {
+ dev_dbg(&twl->client->dev,
+ "can't add keypad data, %d\n",
+ status);
+ platform_device_put(pdev);
+ goto err;
+ }
+ status = platform_device_add(pdev);
+ if (status < 0) {
+ platform_device_put(pdev);
+ dev_dbg(&twl->client->dev,
+ "can't create keypad dev, %d\n",
+ status);
+ goto err;
+ } else {
+ pr_debug("%s: can't alloc keypad dev\n", DRIVER_NAME);
+ status = -ENOMEM;
+ goto err;
+ }
+ }
+
+ if (twl_has_madc() && pdata->macd) {
+ pdev = platform_device_alloc("twl4030_madc", -1);
+ if (pdev) {
+ twl = &twl4030_modules[2];
+ pdev->dev.parent = &twl->client->dev;
+ device_init_wakeup(&pdev->dev, 1);
+ status = platform_device_add_data(pdev, pdata->macd,
+ sizeof(*pdata->macd));
+ if (status < 0) {
+ platform_device_put(pdev);
```

```
+ dev_dbg(&twl->client->dev,  
+ "can't add macd data, %d\n",  
+ status);  
+ goto err;  
+ }  
+ status = platform_device_add(pdev);  
+ if (status < 0) {  
+ platform_device_put(pdev);  
+ dev_dbg(&twl->client->dev,  
+ "can't create macd dev, %d\n",  
+ status);  
+ goto err;  
+ }  
+ } else {  
+ pr_debug("%s: can't alloc macd dev\n", DRIVER_NAME);  
+ status = -ENOMEM;  
+ goto err;  
+ }  
+ }  
+  
+ if (twl_has_rtc()) {  
+ twl = &twl4030_modules[3];  
+  
+ pdev = platform_device_alloc("twl4030_rtc", -1);  
+ if (!pdev) {  
+ pr_debug("%s: can't alloc rtc dev\n", DRIVER_NAME);  
+ status = -ENOMEM;  
+ } else {  
+ pdev->dev.parent = &twl->client->dev;  
+ device_init_wakeup(&pdev->dev, 1);  
+ }  
+  
+ /*  
+ * REVISIT platform_data here currently might use of  
+ * "msecure" line ... but for now we just expect board  
+ * setup to tell the chip "we are secure" at all times.  
+ * Eventually, Linux might become more aware of such  
+ * HW security concerns, and "least privilege".  
+ */  
+  
+ /* RTC module IRQ */  
+ if (status == 0) {  
+ struct resource r = {  
+ /* REVISIT don't hard-wire this stuff */  
+ .start = TWL4030_PWRIRQ_RTC,  
+ .flags = IORESOURCE_IRQ,  
+ };  
+  
+ status = platform_device_add_resources(pdev, &r, 1);  
+ }  
+  
+ }
```

```
+ if (status == 0)
+ status = platform_device_add(pdev);
+
+ if (status < 0) {
+ platform_device_put(pdev);
+ dev_dbg(&twl->client->dev,
+ "can't create rtc dev, %d\n",
+ status);
+ goto err;
+ }
+ }
+
+ if (twl_has_usb() && pdata->usb) {
+ twl = &twl4030_modules[0];
+
+ pdev = platform_device_alloc("twl4030_usb", -1);
+ if (!pdev) {
+ pr_debug("%s: can't alloc usb dev\n", DRIVER_NAME);
+ status = -ENOMEM;
+ goto err;
+ }
+
+ if (status == 0) {
+ pdev->dev.parent = &twl->client->dev;
+ device_init_wakeup(&pdev->dev, 1);
+ status = platform_device_add_data(pdev, pdata->usb,
+ sizeof(*pdata->usb));
+ if (status < 0) {
+ platform_device_put(pdev);
+ dev_dbg(&twl->client->dev,
+ "can't add usb data, %d\n",
+ status);
+ goto err;
+ }
+ }
+
+ if (status == 0) {
+ struct resource r = {
+ .start = TWL4030_PWRIRQ_USB_PRESENCE,
+ .flags = IORESOURCE_IRQ,
+ };
+
+ status = platform_device_add_resources(pdev, &r, 1);
+ }
+
+ if (status == 0)
+ status = platform_device_add(pdev);
+
+ if (status < 0) {
+ platform_device_put(pdev);
+ dev_dbg(&twl->client->dev,
```

```
+ "can't create usb dev, %d\n",
+ status);
+ }
+ }
+
+err:
+ if (status)
+ pr_err("failed to add twl4030's children (status %d)\n", status);
+ return status;
+}
+
+/*-----*/
+
+/*
+ * These three functions initialize the on-chip clock framework,
+ * letting it generate the right frequencies for USB, MADC, and
+ * other purposes.
+ */
+static inline int __init protect_pm_master(void)
+{
+ int e = 0;
+
+ e = twl4030_i2c_write_u8(TWL4030_MODULE_PM_MASTER, KEY_LOCK,
+ R_PROTECT_KEY);
+ return e;
+}
+
+static inline int __init unprotect_pm_master(void)
+{
+ int e = 0;
+
+ e |= twl4030_i2c_write_u8(TWL4030_MODULE_PM_MASTER, KEY_UNLOCK1,
+ R_PROTECT_KEY);
+ e |= twl4030_i2c_write_u8(TWL4030_MODULE_PM_MASTER, KEY_UNLOCK2,
+ R_PROTECT_KEY);
+ return e;
+}
+
+static void __init clocks_init(void)
+{
+ int e = 0;
+ struct clk *osc;
+ u32 rate;
+ u8 ctrl = HFCLK_FREQ_26_MHZ;
+
+#if defined(CONFIG_ARCH_OMAP2) || defined(CONFIG_ARCH_OMAP3)
+ if (cpu_is_omap2430())
+ osc = clk_get(NULL, "osc_ck");
+ else
+ osc = clk_get(NULL, "osc_sys_ck");
+#else
```

```

+ /* REVISIT for non-OMAP systems, pass the clock rate from
+ * board init code, using platform_data.
+ */
+ osc = ERR_PTR(-EIO);
+ #endif
+ if (IS_ERR(osc)) {
+ printk(KERN_WARNING "Skipping twl4030 internal clock init and "
+ "using bootloader value (unknown osc rate)\n");
+ return;
+ }
+
+ rate = clk_get_rate(osc);
+ clk_put(osc);
+
+ switch (rate) {
+ case 19200000:
+ ctrl = HFCLK_FREQ_19p2_MHZ;
+ break;
+ case 26000000:
+ ctrl = HFCLK_FREQ_26_MHZ;
+ break;
+ case 38400000:
+ ctrl = HFCLK_FREQ_38p4_MHZ;
+ break;
+ }
+
+ ctrl |= HIGH_PERF_SQ;
+ e |= unprotect_pm_master();
+ /* effect->MADC+USB ck en */
+ e |= twl4030_i2c_write_u8(TWL4030_MODULE_PM_MASTER, ctrl, R_CFG_BOOT);
+ e |= protect_pm_master();
+
+ if (e < 0)
+ pr_err("%s: clock init err [%d]\n", DRIVER_NAME, e);
+ }
+
+ /*-----*/
+
+ /**
+ * twl4030_i2c_clear_isr - clear TWL4030 SIH ISR regs via read + write
+ * @mod_no: TWL4030 module number
+ * @reg: register index to clear
+ * @cor: value of the <module>_SIH_CTRL.COR bit (1 or 0)
+ *
+ * Either reads (cor == 1) or writes (cor == 0) to a TWL4030 interrupt
+ * status register to ensure that any prior interrupts are cleared.
+ * Returns the status from the I2C read operation.
+ */
+ static int __init twl4030_i2c_clear_isr(u8 mod_no, u8 reg, u8 cor)
+ {
+ u8 tmp;

```

```

+
+ return (cor) ? twl4030_i2c_read_u8(mod_no, &tmp, reg) :
+ twl4030_i2c_write_u8(mod_no, 0xff, reg);
+}
+
+/**
+ * twl4030_read_cor_bit – are TWL module ISRs cleared by reads or writes?
+ * @mod_no: TWL4030 module number
+ * @reg: register index to clear
+ *
+ * Returns 1 if the TWL4030 SIH interrupt status registers (ISRs) for
+ * the specified TWL module are cleared by reads, or 0 if cleared by
+ * writes.
+ */
+static int twl4030_read_cor_bit(u8 mod_no, u8 reg)
+{
+ u8 tmp = 0;
+
+ WARN_ON(twl4030_i2c_read_u8(mod_no, &tmp, reg) < 0);
+
+ tmp &= TWL4030_SIH_CTRL_COR_MASK;
+ tmp >>= __ffs(TWL4030_SIH_CTRL_COR_MASK);
+
+ return tmp;
+}
+
+/**
+ * twl4030_mask_clear_intrs – mask and clear all TWL4030 interrupts
+ * @t: pointer to twl4030_mod_iregs array
+ * @t_sz: ARRAY_SIZE(t) (starting at 1)
+ *
+ * Mask all TWL4030 interrupt mask registers (IMRs) and clear all
+ * interrupt status registers (ISRs). No return value, but will WARN if
+ * any I2C operations fail.
+ */
+static void __init twl4030_mask_clear_intrs(const struct twl4030_mod_iregs *t,
+ const u8 t_sz)
+{
+ int i, j;
+
+ /*
+ * N.B. – further efficiency is possible here. Eight I2C
+ * operations on BCI and GPIO modules are avoidable if I2C
+ * burst read/write transactions were implemented. Would
+ * probably save about 1ms of boot time and a small amount of
+ * power.
+ */
+ for (i = 0; i < t_sz; i++) {
+ const struct twl4030_mod_iregs tmr = t[i];
+ int cor;
+

```

```

+ /* Are ISRs cleared by reads or writes? */
+ cor = twl4030_read_cor_bit(tmr.mod_no, tmr.sih_ctrl);
+
+ for (j = 0; j < tmr.reg_cnt; j++) {
+
+ /* Mask interrupts at the TWL4030 */
+ WARN_ON(twl4030_i2c_write_u8(tmr.mod_no, 0xff,
+ tmr.imrs[j]) < 0);
+
+ /* Clear TWL4030 ISRs */
+ WARN_ON(twl4030_i2c_clear_isr(tmr.mod_no,
+ tmr.isrs[j], cor) < 0);
+ }
+ }
+ }
+
+static void twl_init_irq(int irq_num, unsigned irq_base, unsigned irq_end)
+{
+ int i;
+
+ /*
+ * Mask and clear all TWL4030 interrupts since initially we do
+ * not have any TWL4030 module interrupt handlers present
+ */
+ twl4030_mask_clear_intrs(twl4030_mod_regs,
+ ARRAY_SIZE(twl4030_mod_regs));
+
+ twl4030_irq_base = irq_base;
+
+ /* install an irq handler for each of the PIH modules */
+ for (i = irq_base; i < irq_end; i++) {
+ set_irq_chip_and_handler(i, &twl4030_irq_chip,
+ do_twl4030_module_irq);
+ activate_irq(i);
+ }
+
+ /* install an irq handler to demultiplex the TWL4030 interrupt */
+ set_irq_data(irq_num, start_twl4030_irq_thread(irq_num));
+ set_irq_chained_handler(irq_num, do_twl4030_irq);
+ }
+
+/*-----*/
+
+static int twl4030_remove(struct i2c_client *client)
+{
+ unsigned i;
+
+ /* FIXME undo twl_init_irq() */
+ if (twl4030_irq_base) {
+ dev_err(&client->dev, "can't yet clean up IRQs?\n");

```

```

+ return -ENOSYS;
+ }
+
+ for (i = 0; i < TWL4030_NUM_SLAVES; i++) {
+ struct twl4030_client *twl = &twl4030_modules[i];
+
+ if (twl->client && twl->client != client)
+ i2c_unregister_device(twl->client);
+ twl4030_modules[i].client = NULL;
+ }
+ inuse = false;
+ return 0;
+}
+
+/* NOTE: this driver only handles a single twl4030/tps659x0 chip */
+static int
+twl4030_probe(struct i2c_client *client, const struct i2c_device_id *id)
+{
+ int status;
+ unsigned i;
+ struct twl4030_platform_data *pdata = client->dev.platform_data;
+
+ if (!pdata) {
+ dev_dbg(&client->dev, "no platform data?\n");
+ return -EINVAL;
+ }
+
+ if (i2c_check_functionality(client->adapter, I2C_FUNC_I2C) == 0) {
+ dev_dbg(&client->dev, "can't talk I2C?\n");
+ return -EIO;
+ }
+
+ if (inuse || twl4030_irq_base) {
+ dev_dbg(&client->dev, "driver is already in use\n");
+ return -EBUSY;
+ }
+
+ for (i = 0; i < TWL4030_NUM_SLAVES; i++) {
+ struct twl4030_client *twl = &twl4030_modules[i];
+
+ twl->address = client->addr + i;
+ if (i == 0)
+ twl->client = client;
+ else {
+ twl->client = i2c_new_dummy(client->adapter,
+ twl->address);
+ if (!twl->client) {
+ dev_err(&twl->client->dev,
+ "can't attach client %d\n", i);
+ status = -ENOMEM;
+ goto fail;

```

```

+ }
+ strncpy(twl->client->name, id->name,
+ sizeof(twl->client->name));
+ }
+ mutex_init(&twl->xfer_lock);
+ }
+ inuse = true;
+
+ /* setup clock framework */
+ clocks_init();
+
+ /* Maybe init the T2 Interrupt subsystem */
+ if (client->irq
+ && pdata->irq_base
+ && pdata->irq_end > pdata->irq_base) {
+ twl_init_irq(client->irq, pdata->irq_base, pdata->irq_end);
+ dev_info(&client->dev, "IRQ %d chains IRQs %d..%d\n",
+ client->irq, pdata->irq_base, pdata->irq_end - 1);
+ }
+
+ status = add_children(pdata);
+fail:
+ if (status < 0)
+ twl4030_remove(client);
+ return status;
+}
+
+static const struct i2c_device_id twl4030_ids[] = {
+ { "twl4030", 0 }, /* "Triton 2" */
+ { "tps65950", 0 }, /* catalog version of twl4030 */
+ { "tps65930", 0 }, /* fewer LDOs and DACs; no charger */
+ { "tps65920", 0 }, /* fewer LDOs; no codec or charger */
+ { "twl5030", 0 }, /* T2 updated */
+ { /* end of list */ },
+};
+MODULE_DEVICE_TABLE(i2c, twl4030_ids);
+
+ /* One Client Driver , 4 Clients */
+static struct i2c_driver twl4030_driver = {
+ .driver.name = DRIVER_NAME,
+ .id_table = twl4030_ids,
+ .probe = twl4030_probe,
+ .remove = twl4030_remove,
+};
+
+static int __init twl4030_init(void)
+{
+ return i2c_add_driver(&twl4030_driver);
+}
+subsys_initcall(twl4030_init);
+

```

[patch 2.6.27-rc8-git] add drivers/mfd/twl4030-core.c

```
+static void __exit twl4030_exit(void)
+{
+ i2c_del_driver(&twl4030_driver);
+}
+module_exit(twl4030_exit);
+
+MODULE_AUTHOR("Texas Instruments, Inc.");
+MODULE_DESCRIPTION("I2C Core interface for TWL4030");
+MODULE_LICENSE("GPL");
+--- /dev/null
+++ b/include/linux/i2c/twl4030.h
@@ -0,0 +1,339 @@
+/*
+ * twl4030.h - header for TWL4030 PM and audio CODEC device
+ *
+ * Copyright (C) 2005-2006 Texas Instruments, Inc.
+ *
+ * Based on tlv320aic23.c:
+ * Copyright (c) by Kai Svahn <kai.svahn@xxxxxxxxxx>
+ *
+ * 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.
+ *
+ * This program is distributed in the hope that it will be useful,
+ * but WITHOUT ANY WARRANTY; without even the implied warranty of
+ * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
+ * GNU General Public License for more details.
+ *
+ * You should have received a copy of the GNU General Public License
+ * along with this program; if not, write to the Free Software
+ * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
+ */
+
+#ifndef __TWL4030_H_
+#define __TWL4030_H_
+
+/*
+ * Using the twl4030 core we address registers using a pair
+ * { module id, relative register offset }
+ * which that core then maps to the relevant
+ * { i2c slave, absolute register address }
+ *
+ * The module IDs are meaningful only to the twl4030 core code,
+ * which uses them as array indices to look up the first register
+ * address each module uses within a given i2c slave.
+ */
+
+/* Slave 0 (i2c address 0x48) */
```

[patch 2.6.27-rc8-git] add drivers/mfd/twl4030-core.c

```
+ #define TWL4030_MODULE_USB 0x00
+
+ /* Slave 1 (i2c address 0x49) */
+ #define TWL4030_MODULE_AUDIO_VOICE 0x01
+ #define TWL4030_MODULE_GPIO 0x02
+ #define TWL4030_MODULE_INTBR 0x03
+ #define TWL4030_MODULE_PIH 0x04
+ #define TWL4030_MODULE_TEST 0x05
+
+ /* Slave 2 (i2c address 0x4a) */
+ #define TWL4030_MODULE_KEYPAD 0x06
+ #define TWL4030_MODULE_MADC 0x07
+ #define TWL4030_MODULE_INTERRUPTS 0x08
+ #define TWL4030_MODULE_LED 0x09
+ #define TWL4030_MODULE_MAIN_CHARGE 0x0A
+ #define TWL4030_MODULE_PRECHARGE 0x0B
+ #define TWL4030_MODULE_PWM0 0x0C
+ #define TWL4030_MODULE_PWM1 0x0D
+ #define TWL4030_MODULE_PWMA 0x0E
+ #define TWL4030_MODULE_PWMB 0x0F
+
+ /* Slave 3 (i2c address 0x4b) */
+ #define TWL4030_MODULE_BACKUP 0x10
+ #define TWL4030_MODULE_INT 0x11
+ #define TWL4030_MODULE_PM_MASTER 0x12
+ #define TWL4030_MODULE_PM_RECEIVER 0x13
+ #define TWL4030_MODULE_RTC 0x14
+ #define TWL4030_MODULE_SECURED_REG 0x15
+
+ /*
+ * Read and write single 8-bit registers
+ */
+ int twl4030_i2c_write_u8(u8 mod_no, u8 val, u8 reg);
+ int twl4030_i2c_read_u8(u8 mod_no, u8 *val, u8 reg);
+
+ /*
+ * Read and write several 8-bit registers at once.
+ *
+ * IMPORTANT: For twl4030_i2c_write(), allocate num_bytes + 1
+ * for the value, and populate your data starting at offset 1.
+ */
+ int twl4030_i2c_write(u8 mod_no, u8 *value, u8 reg, u8 num_bytes);
+ int twl4030_i2c_read(u8 mod_no, u8 *value, u8 reg, u8 num_bytes);
+
+ /*-----*/
+
+ /*
+ * NOTE: at up to 1024 registers, this is a big chip.
+ *
+ * Avoid putting register declarations in this file, instead of into
+ * a driver-private file, unless some of the registers in a block
```



```
+ #define REG_GPIO_EDR2 0x29
+ #define REG_GPIO_EDR3 0x2A
+ #define REG_GPIO_EDR4 0x2B
+ #define REG_GPIO_EDR5 0x2C
+ #define REG_GPIO_SIH_CTRL 0x2D
+
+ /* Up to 18 signals are available as GPIOs, when their
+  * pins are not assigned to another use (such as ULPI/USB).
+  */
+ #define TWL4030_GPIO_MAX 18
+
+ /*-----*/
+
+ /*
+  * Keypad register offsets (use TWL4030_MODULE_KEYPAD)
+  * ... SIH/interrupt only
+  */
+
+ #define TWL4030_KEYPAD_KEYP_ISR1 0x11
+ #define TWL4030_KEYPAD_KEYP_IMR1 0x12
+ #define TWL4030_KEYPAD_KEYP_ISR2 0x13
+ #define TWL4030_KEYPAD_KEYP_IMR2 0x14
+ #define TWL4030_KEYPAD_KEYP_SIR 0x15 /* test register */
+ #define TWL4030_KEYPAD_KEYP_EDR 0x16
+ #define TWL4030_KEYPAD_KEYP_SIH_CTRL 0x17
+
+ /*-----*/
+
+ /*
+  * Multichannel ADC register offsets (use TWL4030_MODULE_MADC)
+  * ... SIH/interrupt only
+  */
+
+ #define TWL4030_MADC_ISR1 0x61
+ #define TWL4030_MADC_IMR1 0x62
+ #define TWL4030_MADC_ISR2 0x63
+ #define TWL4030_MADC_IMR2 0x64
+ #define TWL4030_MADC_SIR 0x65 /* test register */
+ #define TWL4030_MADC_EDR 0x66
+ #define TWL4030_MADC_SIH_CTRL 0x67
+
+ /*-----*/
+
+ /*
+  * Battery charger register offsets (use TWL4030_MODULE_INTERRUPTS)
+  */
+
+ #define TWL4030_INTERRUPTS_BCIISR1A 0x0
+ #define TWL4030_INTERRUPTS_BCIISR2A 0x1
+ #define TWL4030_INTERRUPTS_BCIIMR1A 0x2
+ #define TWL4030_INTERRUPTS_BCIIMR2A 0x3
```

```

+#define TWL4030_INTERRUPTS_BCIISR1B 0x4
+#define TWL4030_INTERRUPTS_BCIISR2B 0x5
+#define TWL4030_INTERRUPTS_BCIIMR1B 0x6
+#define TWL4030_INTERRUPTS_BCIIMR2B 0x7
+#define TWL4030_INTERRUPTS_BCISIR1 0x8 /* test register */
+#define TWL4030_INTERRUPTS_BCISIR2 0x9 /* test register */
+#define TWL4030_INTERRUPTS_BCIEDR1 0xa
+#define TWL4030_INTERRUPTS_BCIEDR2 0xb
+#define TWL4030_INTERRUPTS_BCIEDR3 0xc
+#define TWL4030_INTERRUPTS_BCISIHCTRL 0xd
+
+/*-----*/
+
+/*
+ * Power Interrupt block register offsets (use TWL4030_MODULE_INT)
+ */
+
+#define TWL4030_INT_PWR_ISR1 0x0
+#define TWL4030_INT_PWR_IMR1 0x1
+#define TWL4030_INT_PWR_ISR2 0x2
+#define TWL4030_INT_PWR_IMR2 0x3
+#define TWL4030_INT_PWR_SIR 0x4 /* test register */
+#define TWL4030_INT_PWR_EDR1 0x5
+#define TWL4030_INT_PWR_EDR2 0x6
+#define TWL4030_INT_PWR_SIH_CTRL 0x7
+
+/*-----*/
+
+struct twl4030_bci_platform_data {
+ int *battery_tmp_tbl;
+ unsigned int tblsize;
+};
+
+/* TWL4030_GPIO_MAX (18) GPIOs, with interrupts */
+struct twl4030_gpio_platform_data {
+ int gpio_base;
+ unsigned irq_base, irq_end;
+
+ /* For gpio-N, bit (1 << N) in "pullups" is set if that pullup
+ * should be enabled. Else, if that bit is set in "pulldowns",
+ * that pulldown is enabled. Don't waste power by letting any
+ * digital inputs float...
+ */
+ u32 pullups;
+ u32 pulldowns;
+
+ int (*setup)(struct device *dev,
+ unsigned gpio, unsigned ngpio);
+ int (*teardown)(struct device *dev,
+ unsigned gpio, unsigned ngpio);
+};

```

```

+
+struct twl4030_madc_platform_data {
+ int irq_line;
+};
+
+struct twl4030_keypad_data {
+ int rows;
+ int cols;
+ int *keymap;
+ int irq;
+ unsigned int keymapsize;
+ unsigned int rep:1;
+};
+
+enum twl4030_usb_mode {
+ T2_USB_MODE_ULPI = 1,
+ T2_USB_MODE_CEA2011_3PIN = 2,
+};
+
+struct twl4030_usb_data {
+ enum twl4030_usb_mode usb_mode;
+};
+
+struct twl4030_platform_data {
+ unsigned irq_base, irq_end;
+ struct twl4030_bci_platform_data *bci;
+ struct twl4030_gpio_platform_data *gpio;
+ struct twl4030_madc_platform_data *madc;
+ struct twl4030_keypad_data *keypad;
+ struct twl4030_usb_data *usb;
+
+ /* REVISIT more to come ... _nothing_ should be hard-wired */
+};
+
+/*-----*/
+
+/*
+ * FIXME completely stop using TWL4030_IRQ_BASE ... instead, pass the
+ * IRQ data to subsidiary devices using platform device resources.
+ */
+
+/* IRQ information--need base */
+#include <mach/irqs.h>
+/* TWL4030 interrupts */
+
+/* #define TWL4030_MODIRQ_GPIO (TWL4030_IRQ_BASE + 0) */
+#define TWL4030_MODIRQ_KEYPAD (TWL4030_IRQ_BASE + 1)
+#define TWL4030_MODIRQ_BCI (TWL4030_IRQ_BASE + 2)
+#define TWL4030_MODIRQ_MADC (TWL4030_IRQ_BASE + 3)
+/* #define TWL4030_MODIRQ_USB (TWL4030_IRQ_BASE + 4) */
+#define TWL4030_MODIRQ_PWR (TWL4030_IRQ_BASE + 5)

```

[patch 2.6.27-rc8-git] add drivers/mfd/twl4030-core.c

```
+
+#define TWL4030_PWRIRQ_PWRBTN (TWL4030_PWR_IRQ_BASE + 0)
+#define TWL4030_PWRIRQ_CHG_PRES (TWL4030_PWR_IRQ_BASE + 1)
+#define TWL4030_PWRIRQ_USB_PRES (TWL4030_PWR_IRQ_BASE + 2)
+#define TWL4030_PWRIRQ_RTC (TWL4030_PWR_IRQ_BASE + 3)
+#define TWL4030_PWRIRQ_HOT_DIE (TWL4030_PWR_IRQ_BASE + 4)
+#define TWL4030_PWRIRQ_PWROK_TIMEOUT (TWL4030_PWR_IRQ_BASE + 5)
+#define TWL4030_PWRIRQ_MBCHG (TWL4030_PWR_IRQ_BASE + 6)
+#define TWL4030_PWRIRQ_SC_DETECT (TWL4030_PWR_IRQ_BASE + 7)
+
+/* Rest are unused currently*/
+
+/* Offsets to Power Registers */
+#define TWL4030_VDAC_DEV_GRP 0x3B
+#define TWL4030_VDAC_DEDICATED 0x3E
+#define TWL4030_VAUX1_DEV_GRP 0x17
+#define TWL4030_VAUX1_DEDICATED 0x1A
+#define TWL4030_VAUX2_DEV_GRP 0x1B
+#define TWL4030_VAUX2_DEDICATED 0x1E
+#define TWL4030_VAUX3_DEV_GRP 0x1F
+#define TWL4030_VAUX3_DEDICATED 0x22
+
+/* TWL4030 GPIO interrupt definitions */
+
+#define TWL4030_GPIO_IRQ_NO(n) (TWL4030_GPIO_IRQ_BASE + (n))
+#define TWL4030_GPIO_IS_ENABLE 1
+
+/*
+ * Exported TWL4030 GPIO APIs
+ *
+ * WARNING --- use standard GPIO and IRQ calls instead; these will vanish.
+ */
+int twl4030_get_gpio_datain(int gpio);
+int twl4030_request_gpio(int gpio);
+int twl4030_set_gpio_debounce(int gpio, int enable);
+int twl4030_free_gpio(int gpio);
+
+#if defined(CONFIG_TWL4030_BCI_BATTERY) || \
+ defined(CONFIG_TWL4030_BCI_BATTERY_MODULE)
+extern int twl4030charger_usb_en(int enable);
+#else
+static inline int twl4030charger_usb_en(int enable) { return 0; }
+#endif
+
+##endif /* End of __TWL4030_H */
---
```

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/>

[patch 2.6.27-rc8-git] add drivers/mfd/twl4030-core.c