

[PATCH 2.6] Altix serial driver

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2004-06/5532.html>

From: Pat Gefre (pfg_at_sgi.com)

Date: 06/23/04

Date: Wed, 23 Jun 2004 09:48:43 -0500 (CDT)

To: akpm@osdl.org

I'm resending with the signed-off line.... sorry I forgot it.

2.6 patch for our console driver. We converted the driver to use the serial core functions. Also some changes to use sysfs/udev and a different major number.

-- Pat

Patrick Gefre

Silicon Graphics, Inc. (E-Mail) pfg@sgi.com

2750 Blue Water Rd (Voice) (651) 683-3127

Eagan, MN 55121-1400 (FAX) (651) 683-3054

Signed-off-by: Patrick Gefre <pfg@sgi.com>

```
# This is a BitKeeper generated diff -Nru style patch.
#
# ChangeSet
# 2004/06/18 17:06:40-05:00 erikj@attica.americas.sgi.com
# Makefile:
# Remove old sn_serial console driver from makefile
#
# drivers/char/Makefile
# 2004/06/18 17:06:14-05:00 erikj@attica.americas.sgi.com +0 -1
# Remove old sn_serial console driver from makefile
#
diff -Nru a/drivers/char/Makefile b/drivers/char/Makefile
--- a/drivers/char/Makefile 2004-06-18 17:10:28 -05:00
+++ b/drivers/char/Makefile 2004-06-18 17:10:28 -05:00
@@ -41,7 +41,6 @@
obj-$(CONFIG_RIO) += rio/ generic_serial.o
obj-$(CONFIG_HVC_CONSOLE) += hvc_console.o
obj-$(CONFIG_RAW_DRIVER) += raw.o
-obj-$(CONFIG_SGI_L1_SERIAL) += sn_serial.o
obj-$(CONFIG_VIOCONS) += viocons.o
obj-$(CONFIG_VIOTAPE) += viotape.o
```

Linux-Kernel: [PATCH 2.6] Altix serial driver

```
# This is a BitKeeper generated diff -Nru style patch.
#
# ChangeSet
# 2004/06/18 17:01:46-05:00 erikj@attica.americas.sgi.com
# Makefile:
# Adds new sn_console driver to the Makefile
# Kconfig:
# Adds new sn_console driver to Kconfig
# Remove old sn_serial driver from config
# sn_console.c:
# Implementation for new Altix console driver.
# .del-sn_serial.c~9c52f144ac55b7cb:
# Delete: drivers/char/sn_serial.c (removes old driver)
#
# drivers/serial/Makefile
# 2004/06/18 16:55:50-05:00 erikj@attica.americas.sgi.com +1 -0
# Adds new sn_console driver to the Makefile
#
# drivers/serial/Kconfig
# 2004/06/18 16:55:33-05:00 erikj@attica.americas.sgi.com +8 -1
# Adds new sn_console driver to Kconfig
#
# drivers/char/Kconfig
# 2004/06/18 16:54:35-05:00 erikj@attica.americas.sgi.com +0 -16
# Remove old sn_serial driver from config
#
# drivers/serial/sn_console.c
# 2004/06/18 16:32:27-05:00 erikj@attica.americas.sgi.com +1138 -0
#
# drivers/serial/sn_console.c
# 2004/06/18 16:32:27-05:00 erikj@attica.americas.sgi.com +0 -0
# BitKeeper file /data/lwork/attica3/erikj/linux-2.5-console/drivers/serial/sn_console.c
#
# BitKeeper/deleted/.del-sn_serial.c~9c52f144ac55b7cb
# 2004/06/18 16:27:08-05:00 erikj@attica.americas.sgi.com +0 -0
# Delete: drivers/char/sn_serial.c
#
diff -Nru a/drivers/char/Kconfig b/drivers/char/Kconfig
--- a/drivers/char/Kconfig 2004-06-18 17:16:46 -05:00
+++ b/drivers/char/Kconfig 2004-06-18 17:16:46 -05:00
@@ -371,22 +371,6 @@
     If you have an Alchemy AU1000 processor (MIPS based) and you want
     to use serial ports, say Y. Otherwise, say N.

-config SGI_L1_SERIAL
-bool "SGI Altix L1 serial support"
-depends on SERIAL_NONSTANDARD && IA64
-help
-If you have an SGI Altix and you want to use the serial port
-connected to the system controller (you want this!), say Y.
```

Linux-Kernel: [PATCH 2.6] Altix serial driver

```
- Otherwise, say N.
-
-config SGI_L1_SERIAL_CONSOLE
- bool "SGI Altix L1 serial console support"
- depends on SGI_L1_SERIAL
- help
- If you have an SGI Altix and you would like to use the system
- controller serial port as your console (you want this!),
- say Y. Otherwise, say N.
-
config AU1000_SERIAL_CONSOLE
    bool "Enable Au1000 serial console"
    depends on AU1000_UART
diff -Nru a/drivers/char/sn_serial.c b/drivers/char/sn_serial.c
--- a/drivers/char/sn_serial.c 2004-06-18 17:16:46 -05:00
+++ /dev/null Wed Dec 31 16:00:00 196900
@@ -1,1028 +0,0 @@
-/*
- * C-Brick Serial Port (and console) driver for SGI Altix machines.
- *
- * This driver is NOT suitable for talking to the I1-controller for
- * anything other than 'console activities' ---- please use the I1
- * driver for that.
- *
- * This file is subject to the terms and conditions of the GNU General Public
- * License. See the file "COPYING" in the main directory of this archive
- * for more details.
- *
- * Copyright (C) 2003 Silicon Graphics, Inc. All rights reserved.
- */
-
-#include <linux/config.h>
-#include <linux/interrupt.h>
-#include <linux/tty.h>
-#include <linux/serial.h>
-#include <linux/console.h>
-#include <linux/module.h>
-#include <linux/sysrq.h>
-#include <linux/circ_buf.h>
-#include <linux/serial_reg.h>
-#include <asm/uaccess.h>
-#include <asm/sn/sgi.h>
-#include <asm/sn/sn_sal.h>
-#include <asm/sn/pci/pciio.h>
-#include <asm/sn/simulator.h>
-#include <asm/sn/sn2/sn_private.h>
-
-#if defined(CONFIG_SGI_L1_SERIAL_CONSOLE) && defined(CONFIG_MAGIC_SYSRQ)
-static char sysrq_serial_str[] = "\eSYS";
-static char *sysrq_serial_ptr = sysrq_serial_str;
-static unsigned long sysrq_requested;
```

Linux-Kernel: [PATCH 2.6] Altix serial driver

```
–#endif /* CONFIG_SGI_L1_SERIAL_CONSOLE && CONFIG_MAGIC_SYSRQ */
–
–/* minor device number */
–#define SN_SAL_MINOR 64
–
–/* number of characters left in xmit buffer before we ask for more */
–#define WAKEUP_CHARS 128
–
–/* number of characters we can transmit to the SAL console at a time */
–#define SN_SAL_MAX_CHARS 120
–
–#define SN_SAL_EVENT_WRITE_WAKEUP 0
–
–/* 64K, when we're asynch, it must be at least printk's LOG_BUF_LEN to
– * avoid losing chars, (always has to be a power of 2) */
–#define SN_SAL_BUFFER_SIZE (64 * (1 << 10))
–
–#define SN_SAL_UART_FIFO_DEPTH 16
–#define SN_SAL_UART_FIFO_SPEED_CPS 9600/10
–
–/* we don't kmalloc/get_free_page these as we want them available
– * before either of those are initialized */
–static char sn_xmit_buff_mem[SN_SAL_BUFFER_SIZE];
–
–struct volatile_circ_buf {
– char *cb_buf;
– int cb_head;
– int cb_tail;
–};
–
–static struct volatile_circ_buf xmit = { .cb_buf = sn_xmit_buff_mem };
–static char sn_tmp_buffer[SN_SAL_BUFFER_SIZE];
–
–static struct tty_struct *sn_sal_tty;
–
–static struct timer_list sn_sal_timer;
–static int sn_sal_event; /* event type for task queue */
–
–static int sn_sal_is_asynch;
–static int sn_sal_irq;
–static spinlock_t sn_sal_lock = SPIN_LOCK_UNLOCKED;
–static int sn_total_tx_count;
–static int sn_total_rx_count;
–
–static void sn_sal_tasklet_action(unsigned long data);
–static DECLARE_TASKLET(sn_sal_tasklet, sn_sal_tasklet_action, 0);
–
–static unsigned long sn_interrupt_timeout;
–
–extern u64 master_node_bedrock_address;
–
```

```

-#undef DEBUG
-#ifdef DEBUG
-static int sn_debug_printf(const char *fmt, ...);
-#define DPRINTF(x...) sn_debug_printf(x)
-#else
-#define DPRINTF(x...) do { } while (0)
-#endif
-
-struct sn_sal_ops {
- int (*sal_puts)(const char *s, int len);
- int (*sal_getc)(void);
- int (*sal_input_pending)(void);
- void (*sal_wakeup_transmit)(void);
-};
-
-/* This is the pointer used. It is assigned to point to one of
- * the tables below.
- */
-static struct sn_sal_ops *sn_func;
-
-/* Prototypes */
-static int snt_hw_puts(const char *, int);
-static int snt_poll_getc(void);
-static int snt_poll_input_pending(void);
-static int snt_sim_puts(const char *, int);
-static int snt_sim_getc(void);
-static int snt_sim_input_pending(void);
-static int snt_intr_getc(void);
-static int snt_intr_input_pending(void);
-static void sn_intr_transmit_chars(void);
-
-/* A table for polling */
-static struct sn_sal_ops poll_ops = {
- .sal_puts = snt_hw_puts,
- .sal_getc = snt_poll_getc,
- .sal_input_pending = snt_poll_input_pending
-};
-
-/* A table for the simulator */
-static struct sn_sal_ops sim_ops = {
- .sal_puts = snt_sim_puts,
- .sal_getc = snt_sim_getc,
- .sal_input_pending = snt_sim_input_pending
-};
-
-/* A table for interrupts enabled */
-static struct sn_sal_ops intr_ops = {
- .sal_puts = snt_hw_puts,
- .sal_getc = snt_intr_getc,
- .sal_input_pending = snt_intr_input_pending,
- .sal_wakeup_transmit = sn_intr_transmit_chars

```

Linux-Kernel: [PATCH 2.6] Altix serial driver

```
-};  
-  
-  
-/* the console does output in two distinctly different ways:  
- * synchronous and asynchronous (buffered). initially, early_printk  
- * does synchronous output. any data written goes directly to the SAL  
- * to be output (incidentally, it is internally buffered by the SAL)  
- * after interrupts and timers are initialized and available for use,  
- * the console init code switches to asynchronous output. this is  
- * also the earliest opportunity to begin polling for console input.  
- * after console initialization, console output and tty (serial port)  
- * output is buffered and sent to the SAL asynchronously (either by  
- * timer callback or by UART interrupt) */  
-  
-  
-/* routines for running the console in polling mode */  
-  
-static int  
-snt_hw_puts(const char *s, int len)  
-  
-{  
- /* looking at the PROM source code, putb calls the flush  
- * routine, so if we send characters in FIFO sized chunks, it  
- * should go out by the next time the timer gets called */  
- return ia64_sn_console_putb(s, len);  
-}  
-  
-static int  
-snt_poll_getc(void)  
-  
-{  
- int ch;  
- ia64_sn_console_getc(&ch);  
- return ch;  
-}  
-  
-static int  
-snt_poll_input_pending(void)  
-  
-{  
- int status, input;  
-  
- status = ia64_sn_console_check(&input);  
- return !status && input;  
-}  
-  
-  
-/* routines for running the console on the simulator */  
-  
-static int  
-snt_sim_puts(const char *str, int count)  
-  
-{  
- int counter = count;  
-  
-
```

Linux-Kernel: [PATCH 2.6] Altix serial driver

```

-#ifdef FLAG_DIRECT_CONSOLE_WRITES
- /* This is an easy way to pre-pend the output to know whether the output
- * was done via sal or directly */
- writeb('[', master_node_bedrock_address + (UART_TX << 3));
- writeb('+', master_node_bedrock_address + (UART_TX << 3));
- writeb(']', master_node_bedrock_address + (UART_TX << 3));
- writeb(' ', master_node_bedrock_address + (UART_TX << 3));
-#endif /* FLAG_DIRECT_CONSOLE_WRITES */
- while (counter > 0) {
- writeb(*str, master_node_bedrock_address + (UART_TX << 3));
- counter--;
- str++;
- }
-
- return count;
-}
-
-#static int
-#snt_sim_getc(void)
-#{
- return readb(master_node_bedrock_address + (UART_RX << 3));
-}
-
-#static int
-#snt_sim_input_pending(void)
-#{
- return readb(master_node_bedrock_address + (UART_LSR << 3)) & UART_LSR_DR;
-}
-
-/* routines for an interrupt driven console (normal) */
-
-#static int
-#snt_intr_getc(void)
-#{
- return ia64_sn_console_readc();
-}
-
-#static int
-#snt_intr_input_pending(void)
-#{
- return ia64_sn_console_intr_status() & SAL_CONSOLE_INTR_RECV;
-}
-
-/* The early printk (possible setup) and function call */
-
-#void
-#early_printk_sn_sal(const char *s, unsigned count)
-#{
- extern void early_sn_setup(void);
-}
-

```

Linux-Kernel: [PATCH 2.6] Altix serial driver

```
- if (!sn_func) {
- if (IS_RUNNING_ON_SIMULATOR())
- sn_func = &sim_ops;
- else
- sn_func = &poll_ops;
-
- early_sn_setup();
- }
- sn_func->sal_puts(s, count);
-}
-
-#ifdef DEBUG
-/* this is as "close to the metal" as we can get, used when the driver
- * itself may be broken */
-static int
-sn_debug_printf(const char *fmt, ...)
-{
- static char printk_buf[1024];
- int printed_len;
- va_list args;
-
- va_start(args, fmt);
- printed_len = vsnprintf(printk_buf, sizeof(printk_buf), fmt, args);
- early_printk_sn_sal(printk_buf, printed_len);
- va_end(args);
- return printed_len;
-}
-#endif /* DEBUG */
-
-/*
- * Interrupt handling routines.
- */
-
-static void
-sn_sal_sched_event(int event)
-{
- sn_sal_event |= (1 << event);
- tasklet_schedule(&sn_sal_tasklet);
-}
-
-/* sn_receive_chars can be called before sn_sal_tty is initialized. in
- * that case, its only use is to trigger sysrq and kdb */
-static void
-sn_receive_chars(struct pt_regs *regs, unsigned long *flags)
-{
- int ch;
-
- while (sn_func->sal_input_pending()) {
- ch = sn_func->sal_getc();
- if (ch < 0) {
- printk(KERN_ERR "sn_serial: An error occured while "
```

Linux-Kernel: [PATCH 2.6] Altix serial driver

```
- "obtaining data from the console (0x%0x)\n", ch);
- break;
- }
-#if defined(CONFIG_SGI_L1_SERIAL_CONSOLE) && defined(CONFIG_MAGIC_SYSRQ)
- if (sysrq_requested) {
- unsigned long sysrq_timeout = sysrq_requested + HZ*5;
-
- sysrq_requested = 0;
- if (ch && time_before(jiffies, sysrq_timeout)) {
- spin_unlock_irqrestore(&sn_sal_lock, *flags);
- handle_sysrq(ch, regs, NULL);
- spin_lock_irqsave(&sn_sal_lock, *flags);
- /* don't record this char */
- continue;
- }
- }
- if (ch == *sysrq_serial_ptr) {
- if (!(*++sysrq_serial_ptr)) {
- sysrq_requested = jiffies;
- sysrq_serial_ptr = sysrq_serial_str;
- }
- }
- else
- sysrq_serial_ptr = sysrq_serial_str;
-#endif /* CONFIG_SGI_L1_SERIAL_CONSOLE && CONFIG_MAGIC_SYSRQ */
-
- /* record the character to pass up to the tty layer */
- if (sn_sal_tty) {
- *sn_sal_tty->flip.char_buf_ptr = ch;
- sn_sal_tty->flip.char_buf_ptr++;
- sn_sal_tty->flip.count++;
- if (sn_sal_tty->flip.count == TTY_FLIPBUF_SIZE)
- break;
- }
- sn_total_rx_count++;
- }
-
- if (sn_sal_tty)
- tty_flip_buffer_push((struct tty_struct *)sn_sal_tty);
- }
-
- /* synch_flush_xmit must be called with sn_sal_lock */
-static void
- synch_flush_xmit(void)
- {
- int xmit_count, tail, head, loops, ii;
- int result;
- char *start;
-
- if (xmit.cb_head == xmit.cb_tail)
```

Linux-Kernel: [PATCH 2.6] Altix serial driver

```

- return; /* Nothing to do. */
-
- head = xmit.cb_head;
- tail = xmit.cb_tail;
- start = &xmit.cb_buf[tail];
-
- /* twice around gets the tail to the end of the buffer and
- * then to the head, if needed */
- loops = (head < tail) ? 2 : 1;
-
- for (ii = 0; ii < loops; ii++) {
- xmit_count = (head < tail) ? (SN_SAL_BUFFER_SIZE - tail) : (head - tail);
-
- if (xmit_count > 0) {
- result = sn_func->sal_puts((char *)start, xmit_count);
- if (!result)
- DPRINTF("\n*** synch_flush_xmit failed to flush\n");
- if (result > 0) {
- xmit_count -= result;
- sn_total_tx_count += result;
- tail += result;
- tail &= SN_SAL_BUFFER_SIZE - 1;
- xmit.cb_tail = tail;
- start = (char *)&xmit.cb_buf[tail];
- }
- }
- }
- }
-
- /* must be called with a lock protecting the circular buffer and
- * sn_sal_tty */
-static void
-sn_poll_transmit_chars(void)
-{
- int xmit_count, tail, head;
- int result;
- char *start;
-
- BUG_ON(!sn_sal_is_async);
-
- if (xmit.cb_head == xmit.cb_tail ||
- (sn_sal_tty && (sn_sal_tty->stopped || sn_sal_tty->hw_stopped))) {
- /* Nothing to do. */
- return;
- }
-
- head = xmit.cb_head;
- tail = xmit.cb_tail;
- start = &xmit.cb_buf[tail];
-
- xmit_count = (head < tail) ? (SN_SAL_BUFFER_SIZE - tail) : (head - tail);

```

Linux-Kernel: [PATCH 2.6] Altix serial driver

```

-
- if (xmit_count == 0)
- DPRINTF("\n*** empty xmit_count\n");
-
- /* use the ops, as we could be on the simulator */
- result = sn_func->sal_puts((char *)start, xmit_count);
- if (!result)
- DPRINTF("\n*** error in synchronous sal_puts\n");
- /* XXX chadt clean this up */
- if (result > 0) {
- xmit_count -= result;
- sn_total_tx_count += result;
- tail += result;
- tail &= SN_SAL_BUFFER_SIZE - 1;
- xmit.cb_tail = tail;
- start = &xmit.cb_buf[tail];
- }
-
- /* if there's few enough characters left in the xmit buffer
- * that we could stand for the upper layer to send us some
- * more, ask for it. */
- if (sn_sal_tty)
- if (CIRC_CNT(xmit.cb_head, xmit.cb_tail, SN_SAL_BUFFER_SIZE) < WAKEUP_CHARS)
- sn_sal_sched_event(SN_SAL_EVENT_WRITE_WAKEUP);
- }
-
-
- /* must be called with a lock protecting the circular buffer and
- * sn_sal_tty */
- static void
- sn_intr_transmit_chars(void)
- {
- int xmit_count, tail, head, loops, ii;
- int result;
- char *start;
-
- BUG_ON(!sn_sal_is_async);
-
- if (xmit.cb_head == xmit.cb_tail ||
- (sn_sal_tty && (sn_sal_tty->stopped || sn_sal_tty->hw_stopped))) {
- /* Nothing to do. */
- return;
- }
-
- head = xmit.cb_head;
- tail = xmit.cb_tail;
- start = &xmit.cb_buf[tail];
-
- /* twice around gets the tail to the end of the buffer and
- * then to the head, if needed */
- loops = (head < tail) ? 2 : 1;

```

```

-
- for (ii = 0; ii < loops; ii++) {
- xmit_count = (head < tail) ?
- (SN_SAL_BUFFER_SIZE - tail) : (head - tail);
-
- if (xmit_count > 0) {
- result = ia64_sn_console_xmit_chars((char *)start, xmit_count);
- #ifdef DEBUG
- if (!result)
- DPRINTF("\n");
- #endif
- if (result > 0) {
- xmit_count -= result;
- sn_total_tx_count += result;
- tail += result;
- tail &= SN_SAL_BUFFER_SIZE - 1;
- xmit.cb_tail = tail;
- start = &xmit.cb_buf[tail];
- }
- }
- }
-
- /* if there's few enough characters left in the xmit buffer
- * that we could stand for the upper layer to send us some
- * more, ask for it. */
- if (sn_sal_tty)
- if (CIRC_CNT(xmit.cb_head, xmit.cb_tail, SN_SAL_BUFFER_SIZE) < WAKEUP_CHARS)
- sn_sal_sched_event(SN_SAL_EVENT_WRITE_WAKEUP);
- }
-
-
- static irqreturn_t
- sn_sal_interrupt(int irq, void *dev_id, struct pt_regs *regs)
- {
- /* this call is necessary to pass the interrupt back to the
- * SAL, since it doesn't intercept the UART interrupts
- * itself */
- int status = ia64_sn_console_intr_status();
- unsigned long flags;
-
- spin_lock_irqsave(&sn_sal_lock, flags);
- if (status & SAL_CONSOLE_INTR_RECV)
- sn_receive_chars(regs, &flags);
- if (status & SAL_CONSOLE_INTR_XMIT)
- sn_intr_transmit_chars();
- spin_unlock_irqrestore(&sn_sal_lock, flags);
- return IRQ_HANDLED;
- }
-
-
- /* returns the console irq if interrupt is successfully registered,

```

Linux-Kernel: [PATCH 2.6] Altix serial driver

```
- * else 0 */
-static int
-sn_sal_connect_interrupt(void)
-{
- cpuid_t intr_cpuid;
- unsigned int intr_cpuloc;
- nasid_t console_nasid;
- unsigned int console_irq;
- int result;
-
- console_nasid = ia64_sn_get_console_nasid();
- intr_cpuid = first_cpu(node_to_cpumask(nasid_to_cnodeid(console_nasid)));
- intr_cpuloc = cpu_physical_id(intr_cpuid);
- console_irq = CPU_VECTOR_TO_IRQ(intr_cpuloc, SGI_UART_VECTOR);
-
- result = intr_connect_level(intr_cpuid, SGI_UART_VECTOR);
- BUG_ON(result != SGI_UART_VECTOR);
-
- result = request_irq(console_irq, sn_sal_interrupt, SA_INTERRUPT, "SAL console driver", &sn_sal_tty);
- if (result >= 0)
- return console_irq;
-
- printk(KERN_WARNING "sn_serial: console proceeding in polled mode\n");
- return 0;
-}
-
-static void
-sn_sal_tasklet_action(unsigned long data)
-{
- unsigned long flags;
-
- if (sn_sal_tty) {
- spin_lock_irqsave(&sn_sal_lock, flags);
- if (sn_sal_tty) {
- if (test_and_clear_bit(SN_SAL_EVENT_WRITE_WAKEUP, &sn_sal_event)) {
- if ((sn_sal_tty->flags & (1 << TTY_DO_WRITE_WAKEUP)) && sn_sal_tty->ldisc.write_wakeup)
- (sn_sal_tty->ldisc.write_wakeup)((struct tty_struct *)sn_sal_tty);
- wake_up_interruptible((wait_queue_head_t *)&sn_sal_tty->write_wait);
- }
- }
- spin_unlock_irqrestore(&sn_sal_lock, flags);
- }
- }
-
-/*
- * This function handles polled mode.
- */
-static void
-sn_sal_timer_poll(unsigned long dummy)
-{
```

Linux-Kernel: [PATCH 2.6] Altix serial driver

```
- unsigned long flags;
-
- if (!sn_sal_irq) {
- spin_lock_irqsave(&sn_sal_lock, flags);
- sn_receive_chars(NULL, &flags);
- sn_poll_transmit_chars();
- spin_unlock_irqrestore(&sn_sal_lock, flags);
- mod_timer(&sn_sal_timer, jiffies + sn_interrupt_timeout);
- }
-}
-
-/*
- * User-level console routines
- */
-
-static int
-sn_sal_open(struct tty_struct *tty, struct file *filp)
-{
- unsigned long flags;
-
- DPRINTF("sn_sal_open: sn_sal_tty = %p, tty = %p, filp = %p\n",
- sn_sal_tty, tty, filp);
-
- spin_lock_irqsave(&sn_sal_lock, flags);
- if (!sn_sal_tty)
- sn_sal_tty = tty;
- spin_unlock_irqrestore(&sn_sal_lock, flags);
-
- return 0;
-}
-
-/* We're keeping all our resources. We're keeping interrupts turned
- * on. Maybe just let the tty layer finish its stuff...? GMSH
- */
-static void
-sn_sal_close(struct tty_struct *tty, struct file * filp)
-{
- if (tty->count == 1) {
- unsigned long flags;
- tty->closing = 1;
- if (tty->driver->flush_buffer)
- tty->driver->flush_buffer(tty);
- if (tty->ldisc.flush_buffer)
- tty->ldisc.flush_buffer(tty);
- tty->closing = 0;
- spin_lock_irqsave(&sn_sal_lock, flags);
- sn_sal_tty = NULL;
- spin_unlock_irqrestore(&sn_sal_lock, flags);
- }
-}
```

Linux-Kernel: [PATCH 2.6] Altix serial driver

```
-}
-
-
-static int
-sn_sal_write(struct tty_struct *tty, int from_user,
-const unsigned char *buf, int count)
-{
- int c, ret = 0;
- unsigned long flags;
-
- if (from_user) {
- while (1) {
- int c1;
- c = CIRC_SPACE_TO_END(xmit.cb_head, xmit.cb_tail,
- SN_SAL_BUFFER_SIZE);
-
- if (count < c)
- c = count;
- if (c <= 0)
- break;
-
- c -= copy_from_user(sn_tmp_buffer, buf, c);
- if (!c) {
- if (!ret)
- ret = -EFAULT;
- break;
- }
-
- /* Turn off interrupts and see if the xmit buffer has
- * moved since the last time we looked.
- */
- spin_lock_irqsave(&sn_sal_lock, flags);
- c1 = CIRC_SPACE_TO_END(xmit.cb_head, xmit.cb_tail, SN_SAL_BUFFER_SIZE);
-
- if (c1 < c)
- c = c1;
-
- memcpy(xmit.cb_buf + xmit.cb_head, sn_tmp_buffer, c);
- xmit.cb_head = ((xmit.cb_head + c) & (SN_SAL_BUFFER_SIZE - 1));
- spin_unlock_irqrestore(&sn_sal_lock, flags);
-
- buf += c;
- count -= c;
- ret += c;
- }
- }
- else {
- /* The buffer passed in isn't coming from userland,
- * so cut out the middleman (sn_tmp_buffer).
- */
- spin_lock_irqsave(&sn_sal_lock, flags);
```

Linux-Kernel: [PATCH 2.6] Altix serial driver

```
- while (1) {
- c = CIRC_SPACE_TO_END(xmit.cb_head, xmit.cb_tail, SN_SAL_BUFFER_SIZE);
-
- if (count < c)
- c = count;
- if (c <= 0) {
- break;
- }
- memcpy(xmit.cb_buf + xmit.cb_head, buf, c);
- xmit.cb_head = ((xmit.cb_head + c) & (SN_SAL_BUFFER_SIZE - 1));
- buf += c;
- count -= c;
- ret += c;
- }
- spin_unlock_irqrestore(&sn_sal_lock, flags);
- }
-
- spin_lock_irqsave(&sn_sal_lock, flags);
- if (xmit.cb_head != xmit.cb_tail && !(tty && (tty->stopped || tty->hw_stopped)))
- if (sn_func->sal_wakeup_transmit)
- sn_func->sal_wakeup_transmit();
- spin_unlock_irqrestore(&sn_sal_lock, flags);
-
- return ret;
-}
-
-
-static void
-sn_sal_put_char(struct tty_struct *tty, unsigned char ch)
-{
- unsigned long flags;
-
- spin_lock_irqsave(&sn_sal_lock, flags);
- if (CIRC_SPACE(xmit.cb_head, xmit.cb_tail, SN_SAL_BUFFER_SIZE) != 0) {
- xmit.cb_buf[xmit.cb_head] = ch;
- xmit.cb_head = (xmit.cb_head + 1) & (SN_SAL_BUFFER_SIZE-1);
- if ( sn_func->sal_wakeup_transmit )
- sn_func->sal_wakeup_transmit();
- }
- spin_unlock_irqrestore(&sn_sal_lock, flags);
-}
-
-
-
-
-static void
-sn_sal_flush_chars(struct tty_struct *tty)
-{
- unsigned long flags;
-
- spin_lock_irqsave(&sn_sal_lock, flags);
- if (CIRC_CNT(xmit.cb_head, xmit.cb_tail, SN_SAL_BUFFER_SIZE))
- if (sn_func->sal_wakeup_transmit)
```

Linux-Kernel: [PATCH 2.6] Altix serial driver

```
- sn_func->sal_wakeup_transmit();
- spin_unlock_irqrestore(&sn_sal_lock, flags);
-}
-
-
-static int
-sal_write_room(struct tty_struct *tty)
-{
- unsigned long flags;
- int space;
-
- spin_lock_irqsave(&sn_sal_lock, flags);
- space = CIRC_SPACE(xmit.cb_head, xmit.cb_tail, SN_SAL_BUFFER_SIZE);
- spin_unlock_irqrestore(&sn_sal_lock, flags);
- return space;
-}
-
-
-static int
-sal_chars_in_buffer(struct tty_struct *tty)
-{
- unsigned long flags;
- int space;
-
- spin_lock_irqsave(&sn_sal_lock, flags);
- space = CIRC_CNT(xmit.cb_head, xmit.cb_tail, SN_SAL_BUFFER_SIZE);
- DPRINTF("<%d>", space);
- spin_unlock_irqrestore(&sn_sal_lock, flags);
- return space;
-}
-
-
-static void
-sal_flush_buffer(struct tty_struct *tty)
-{
- unsigned long flags;
-
- /* drop everything */
- spin_lock_irqsave(&sn_sal_lock, flags);
- xmit.cb_head = xmit.cb_tail = 0;
- spin_unlock_irqrestore(&sn_sal_lock, flags);
-
- /* wake up tty level */
- wake_up_interruptible(&tty->write_wait);
- if ((tty->flags & (1 << TTY_DO_WRITE_WAKEUP)) && tty->ldisc.write_wakeup)
- (tty->ldisc.write_wakeup)(tty);
-}
-
-
-static void
-sal_hangup(struct tty_struct *tty)
```

```

- {
- sn_sal_flush_buffer(tty);
- }
-
-
- static void
- sn_sal_wait_until_sent(struct tty_struct *tty, int timeout)
- {
- /* this is SAL's problem */
- DPRINTF("<sn_serial: should wait until sent>");
- }
-
-
- /*
- * sn_sal_read_proc
- *
- * Console /proc interface
- */
-
- static int
- sn_sal_read_proc(char *page, char **start, off_t off, int count,
- int *eof, void *data)
- {
- int len = 0;
- off_t begin = 0;
-
- len += sprintf(page, "sn_serial: nasid:%ld irq:%d tx:%d rx:%d\n",
- ia64_sn_get_console_nasid(), sn_sal_irq,
- sn_total_tx_count, sn_total_rx_count);
- *eof = 1;
-
- if (off >= len+begin)
- return 0;
- *start = page + (off-begin);
-
- return count < begin+len-off ? count : begin+len-off;
- }
-
-
- static struct tty_operations sn_sal_driver_ops = {
- .open = sn_sal_open,
- .close = sn_sal_close,
- .write = sn_sal_write,
- .put_char = sn_sal_put_char,
- .flush_chars = sn_sal_flush_chars,
- .write_room = sn_sal_write_room,
- .chars_in_buffer = sn_sal_chars_in_buffer,
- .hangup = sn_sal_hangup,
- .wait_until_sent = sn_sal_wait_until_sent,
- .read_proc = sn_sal_read_proc,
- };

```

Linux-Kernel: [PATCH 2.6] Altix serial driver

```
-static struct tty_driver *sn_sal_driver;
-
-/* sn_sal_init wishlist:
- * - allocate sn_tmp_buffer
- * - fix up the tty_driver struct
- * - turn on receive interrupts
- * - do any termios twiddling once and for all
- */
-
-/*
- * Boot-time initialization code
- */
-
-static void __init
-sn_sal_switch_to_asynch(void)
-{
- unsigned long flags;
-
- /* without early_printk, we may be invoked late enough to race
- * with other cpus doing console IO at this point, however
- * console interrupts will never be enabled */
- spin_lock_irqsave(&sn_sal_lock, flags);
-
- if (sn_sal_is_asynch) {
- spin_unlock_irqrestore(&sn_sal_lock, flags);
- return;
- }
-
- DPRINTF("sn_serial: switch to asynchronous console\n");
-
- /* early_printk invocation may have done this for us */
- if (!sn_func) {
- if (IS_RUNNING_ON_SIMULATOR())
- sn_func = &sim_ops;
- else
- sn_func = &poll_ops;
- }
-
- /* we can't turn on the console interrupt (as request_irq
- * calls kmalloc, which isn't set up yet), so we rely on a
- * timer to poll for input and push data from the console
- * buffer.
- */
- init_timer(&sn_sal_timer);
- sn_sal_timer.function = sn_sal_timer_poll;
-
- if (IS_RUNNING_ON_SIMULATOR())
- sn_interrupt_timeout = 6;
- else {
- /* 960cps / 16 char FIFO = 60HZ
- * HZ / (SN_SAL_FIFO_SPEED_CPS / SN_SAL_FIFO_DEPTH) */
```

Linux-Kernel: [PATCH 2.6] Altix serial driver

```
- sn_interrupt_timeout = HZ * SN_SAL_UART_FIFO_DEPTH / SN_SAL_UART_FIFO_SPEED_CPS;
- }
- mod_timer(&sn_sal_timer, jiffies + sn_interrupt_timeout);
-
- sn_sal_is_async = 1;
- spin_unlock_irqrestore(&sn_sal_lock, flags);
-}
-
-static void __init
-sn_sal_switch_to_interrupts(void)
-{
- int irq;
-
- DPRINTF("sn_serial: switching to interrupt driven console\n");
-
- irq = sn_sal_connect_interrupt();
- if (irq) {
- unsigned long flags;
- spin_lock_irqsave(&sn_sal_lock, flags);
-
- /* sn_sal_irq is a global variable. When it's set to
- * a non-zero value, we stop polling for input (since
- * interrupts should now be enabled). */
- sn_sal_irq = irq;
- sn_func = &intr_ops;
-
- /* turn on receive interrupts */
- ia64_sn_console_intr_enable(SAL_CONSOLE_INTR_RECV);
- spin_unlock_irqrestore(&sn_sal_lock, flags);
- }
-}
-
-static int __init
-sn_sal_module_init(void)
-{
- int retval;
-
- DPRINTF("sn_serial: sn_sal_module_init\n");
-
- if (!ia64_platform_is("sn2"))
- return -ENODEV;
-
- sn_sal_driver = alloc_tty_driver(1);
- if (!sn_sal_driver)
- return -ENOMEM;
-
- sn_sal_driver->owner = THIS_MODULE;
- sn_sal_driver->driver_name = "sn_serial";
- sn_sal_driver->name = "ttyS";
- sn_sal_driver->major = TTY_MAJOR;
- sn_sal_driver->minor_start = SN_SAL_MINOR;
```

Linux-Kernel: [PATCH 2.6] Altix serial driver

```
- sn_sal_driver->type = TTY_DRIVER_TYPE_SERIAL;
- sn_sal_driver->subtype = SERIAL_TYPE_NORMAL;
- sn_sal_driver->flags = TTY_DRIVER_REAL_RAW | TTY_DRIVER_NO_DEVFS;
-
- tty_set_operations(sn_sal_driver, &sn_sal_driver_ops);
-
- /* when this driver is compiled in, the console initialization
- * will have already switched us into asynchronous operation
- * before we get here through the module initcalls */
- sn_sal_switch_to_async();
-
- /* at this point (module_init) we can try to turn on interrupts */
- if (!IS_RUNNING_ON_SIMULATOR())
- sn_sal_switch_to_interrupts();
-
- sn_sal_driver->init_termios = tty_std_termios;
- sn_sal_driver->init_termios.c_cflag = B9600 | CS8 | CREAD | HUPCL | CLOCAL;
-
- if ((retval = tty_register_driver(sn_sal_driver))) {
- printk(KERN_ERR "sn_serial: Unable to register tty driver\n");
- return retval;
- }
- return 0;
-}
-
-
- static void __exit
- sn_sal_module_exit(void)
- {
- del_timer_sync(&sn_sal_timer);
- tty_unregister_driver(sn_sal_driver);
- put_tty_driver(sn_sal_driver);
- }
-
- module_init(sn_sal_module_init);
- module_exit(sn_sal_module_exit);
-
- /*
- * Kernel console definitions
- */
-
- #ifdef CONFIG_SGI_L1_SERIAL_CONSOLE
- /*
- * Print a string to the SAL console. The console_lock must be held
- * when we get here.
- */
- static void
- sn_sal_console_write(struct console *co, const char *s, unsigned count)
- {
- unsigned long flags;
- const char *s1;
```

```

-
- BUG_ON(!sn_sal_is_async);
-
- /* somebody really wants this output, might be an
- * oops, kdb, panic, etc. make sure they get it. */
- if (spin_is_locked(&sn_sal_lock)) {
-     synch_flush_xmit();
-     /* Output '\r' before each '\n' */
-     while ((s1 = memchr(s, '\n', count)) != NULL) {
-         sn_func->sal_puts(s, s1 - s);
-         sn_func->sal_puts("\r\n", 2);
-         count -= s1 + 1 - s;
-         s = s1 + 1;
-     }
-     sn_func->sal_puts(s, count);
- }
- else if (in_interrupt()) {
-     spin_lock_irqsave(&sn_sal_lock, flags);
-     synch_flush_xmit();
-     spin_unlock_irqrestore(&sn_sal_lock, flags);
-     /* Output '\r' before each '\n' */
-     while ((s1 = memchr(s, '\n', count)) != NULL) {
-         sn_func->sal_puts(s, s1 - s);
-         sn_func->sal_puts("\r\n", 2);
-         count -= s1 + 1 - s;
-         s = s1 + 1;
-     }
-     sn_func->sal_puts(s, count);
- }
- else {
-     /* Output '\r' before each '\n' */
-     while ((s1 = memchr(s, '\n', count)) != NULL) {
-         sn_sal_write(NULL, 0, s, s1 - s);
-         sn_sal_write(NULL, 0, "\r\n", 2);
-         count -= s1 + 1 - s;
-         s = s1 + 1;
-     }
-     sn_sal_write(NULL, 0, s, count);
- }
-}
-
- static struct tty_driver *
- sn_sal_console_device(struct console *c, int *index)
- {
-     *index = c->index;
-     return sn_sal_driver;
- }
-
- static int __init
- sn_sal_console_setup(struct console *co, char *options)
- {

```

Linux-Kernel: [PATCH 2.6] Altix serial driver

```
- return 0;
-}
-
-
-static struct console sal_console = {
- .name = "ttyS",
- .write = sn_sal_console_write,
- .device = sn_sal_console_device,
- .setup = sn_sal_console_setup,
- .index = -1
-};
-
-static int __init
-sn_sal_serial_console_init(void)
-{
- if (ia64_platform_is("sn2")) {
- sn_sal_switch_to_async();
- DPRINTF("sn_sal_serial_console_init : register console\n");
- register_console(&sal_console);
- }
- return 0;
-}
-console_initcall(sn_sal_serial_console_init);
-
-#endif /* CONFIG_SGI_L1_SERIAL_CONSOLE */
diff -Nru a/drivers/serial/Kconfig b/drivers/serial/Kconfig
--- a/drivers/serial/Kconfig 2004-06-18 17:16:46 -05:00
+++ b/drivers/serial/Kconfig 2004-06-18 17:16:46 -05:00
@@ -603,5 +603,12 @@
     on your PowerMac as the console, you can do so by answering
     Y to this option.

-endmenu
+config SERIAL_SGI_L1_CONSOLE
+bool "SGI Altix L1 serial console support"
+select SERIAL_CORE
+help
+If you have an SGI Altix and you would like to use the system
+controller serial port as your console (you want this!),
+say Y. Otherwise, say N.

+endmenu
diff -Nru a/drivers/serial/Makefile b/drivers/serial/Makefile
--- a/drivers/serial/Makefile 2004-06-18 17:16:46 -05:00
+++ b/drivers/serial/Makefile 2004-06-18 17:16:46 -05:00
@@ -38,3 +38,4 @@
obj-$(CONFIG_SERIAL_DZ) += dz.o
obj-$(CONFIG_SERIAL_SH_SCI) += sh-sci.o
obj-$(CONFIG_SERIAL_BAST_SIO) += bast_sio.o
+obj-$(CONFIG_SERIAL_SGI_L1_CONSOLE) += sn_console.o
diff -Nru a/drivers/serial/sn_console.c b/drivers/serial/sn_console.c
```

Linux-Kernel: [PATCH 2.6] Altix serial driver

```
--- /dev/null Wed Dec 31 16:00:00 196900
+++ b/drivers/serial/sn_console.c 2004-06-18 17:16:46 -05:00
@@ -0,0 +1,1138 @@
+/*
+ * C-Brick Serial Port (and console) driver for SGI Altix machines.
+ *
+ * This driver is NOT suitable for talking to the I1-controller for
+ * anything other than 'console activities' --- please use the I1
+ * driver for that.
+ *
+ *
+ * Copyright (c) 2004 Silicon Graphics, Inc. All Rights Reserved.
+ *
+ * This program is free software; you can redistribute it and/or modify it
+ * under the terms of version 2 of the GNU General Public License
+ * as published by the Free Software Foundation.
+ *
+ * This program is distributed in the hope that it would be useful, but
+ * WITHOUT ANY WARRANTY; without even the implied warranty of
+ * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
+ *
+ * Further, this software is distributed without any warranty that it is
+ * free of the rightful claim of any third person regarding infringement
+ * or the like. Any license provided herein, whether implied or
+ * otherwise, applies only to this software file. Patent licenses, if
+ * any, provided herein do not apply to combinations of this program with
+ * other software, or any other product whatsoever.
+ *
+ * You should have received a copy of the GNU General Public
+ * License along with this program; if not, write the Free Software
+ * Foundation, Inc., 59 Temple Place - Suite 330, Boston MA 02111-1307, USA.
+ *
+ * Contact information: Silicon Graphics, Inc., 1500 Crittenden Lane,
+ * Mountain View, CA 94043, or:
+ *
+ * http://www.sgi.com
+ *
+ * For further information regarding this notice, see:
+ *
+ * http://oss.sgi.com/projects/GenInfo/NoticeExplan
+ */
+
+#include <linux/config.h>
+#include <linux/interrupt.h>
+#include <linux/tty.h>
+#include <linux/serial.h>
+#include <linux/console.h>
+#include <linux/module.h>
+#include <linux/sysrq.h>
+#include <linux/circ_buf.h>
+#include <linux/serial_reg.h>
```

Linux–Kernel: [PATCH 2.6] Altix serial driver

```
+#include <linux/delay.h> /* for mdelay */
+#include <linux/miscdevice.h>
+
+#include <asm/sn/simulator.h>
+#include <asm/sn/sn2/sn_private.h>
+#include <linux/serial_core.h>
+#include <asm/sn/sn_sal.h>
+
+/* number of characters we can transmit to the SAL console at a time */
+#define SN_SAL_MAX_CHARS 120
+
+/* 64K, when we're asynch, it must be at least printk's LOG_BUF_LEN to
+ * avoid losing chars, (always has to be a power of 2) */
+#define SN_SAL_BUFFER_SIZE (64 * (1 << 10))
+
+#define SN_SAL_UART_FIFO_DEPTH 16
+#define SN_SAL_UART_FIFO_SPEED_CPS 9600/10
+
+/* sn_transmit_chars() calling args */
+#define TRANSMIT_BUFFERED 0
+#define TRANSMIT_RAW 1
+
+/* To use sysfs only and not use the assigned major and minor, define
+ * the following.. */
+/* #define SYSFS_ONLY 1 */ /* only use SYSFS */
+#define SYSFS_ONLY 0 /* Don't rely on misc_register dynamic minor */
+
+/* Device name we're using */
+#define DEVICE_NAME "ttySG"
+#define DEVICE_NAME_DYNAMIC "ttySG0" /* need full name for misc_register */
+/* The major/minor we are using, ignored for SYSFS_ONLY */
+#define DEVICE_MAJOR 204
+#define DEVICE_MINOR 40
+
+
+/*
+ * Port definition – this kinda drives it all
+ */
+struct sn_cons_port {
+ struct timer_list sc_timer;
+ struct uart_port sc_port;
+ struct sn_sal_ops {
+ int (*sal_puts_raw) (const char *s, int len);
+ int (*sal_puts) (const char *s, int len);
+ int (*sal_getc) (void);
+ int (*sal_input_pending) (void);
+ void (*sal_wakeup_transmit) (struct sn_cons_port *, int);
+ } *sc_ops;
+ unsigned long sc_interrupt_timeout;
+ int sc_is_asynch;
+};
+
```

```

+static struct sn_cons_port sal_console_port;
+
+/* Only used if SYSFS_ONLY is set to 1 */
+static struct miscdevice misc; /* used with misc_register for dynamic */
+
+extern u64 master_node_bedrock_address;
+
+static int sn_debug_printf(const char *fmt, ...);
+
+#undef DEBUG
+#ifdef DEBUG
+#define DPRINTF(x...) sn_debug_printf(x)
+#else
+#define DPRINTF(x...) do { } while (0)
+#endif
+
+/* Prototypes */
+static int snt_hw_puts_raw(const char *, int);
+static int snt_hw_puts_buffered(const char *, int);
+static int snt_poll_getc(void);
+static int snt_poll_input_pending(void);
+static int snt_sim_puts(const char *, int);
+static int snt_sim_getc(void);
+static int snt_sim_input_pending(void);
+static int snt_intr_getc(void);
+static int snt_intr_input_pending(void);
+static void sn_transmit_chars(struct sn_cons_port *, int);
+
+/* A table for polling:
+ */
+static struct sn_sal_ops poll_ops = {
+ .sal_puts_raw = snt_hw_puts_raw,
+ .sal_puts = snt_hw_puts_buffered,
+ .sal_getc = snt_poll_getc,
+ .sal_input_pending = snt_poll_input_pending
+};
+
+/* A table for the simulator */
+static struct sn_sal_ops sim_ops = {
+ .sal_puts_raw = snt_sim_puts,
+ .sal_puts = snt_sim_puts,
+ .sal_getc = snt_sim_getc,
+ .sal_input_pending = snt_sim_input_pending
+};
+
+/* A table for interrupts enabled */
+static struct sn_sal_ops intr_ops = {
+ .sal_puts_raw = snt_hw_puts_raw,
+ .sal_puts = snt_hw_puts_buffered,
+ .sal_getc = snt_intr_getc,
+ .sal_input_pending = snt_intr_input_pending,

```

```

+ .sal_wakeup_transmit = sn_transmit_chars
+};
+
+/* the console does output in two distinctly different ways:
+ * synchronous (raw) and asynchronous (buffered). initially, early_printk
+ * does synchronous output. any data written goes directly to the SAL
+ * to be output (incidentally, it is internally buffered by the SAL)
+ * after interrupts and timers are initialized and available for use,
+ * the console init code switches to asynchronous output. this is
+ * also the earliest opportunity to begin polling for console input.
+ * after console initialization, console output and tty (serial port)
+ * output is buffered and sent to the SAL asynchronously (either by
+ * timer callback or by UART interrupt) */
+
+
+/* routines for running the console in polling mode */
+
+/**
+ * snt_poll_getc – Get a character from the console in polling mode
+ *
+ */
+static int
+snt_poll_getc(void)
+{
+ int ch;
+
+ ia64_sn_console_getc(&ch);
+ return ch;
+}
+
+/**
+ * snt_poll_input_pending – Check if any input is waiting – polling mode.
+ *
+ */
+static int
+snt_poll_input_pending(void)
+{
+ int status, input;
+
+ status = ia64_sn_console_check(&input);
+ return !status && input;
+}
+
+/* routines for running the console on the simulator */
+
+/**
+ * snt_sim_puts – send to the console, used in simulator mode
+ * @str: String to send
+ * @count: length of string
+ *
+ */

```

```

+static int
+snt_sim_puts(const char *str, int count)
+{
+ int counter = count;
+
+#ifdef FLAG_DIRECT_CONSOLE_WRITES
+ /* This is an easy way to pre-pend the output to know whether the output
+ * was done via sal or directly */
+ writeb('[', master_node_bedrock_address + (UART_TX << 3));
+ writeb('+', master_node_bedrock_address + (UART_TX << 3));
+ writeb(']', master_node_bedrock_address + (UART_TX << 3));
+ writeb(' ', master_node_bedrock_address + (UART_TX << 3));
+#endif /* FLAG_DIRECT_CONSOLE_WRITES */
+ while (counter > 0) {
+ writeb(*str, master_node_bedrock_address + (UART_TX << 3));
+ counter--;
+ str++;
+ }
+ return count;
+}
+
+/**
+ * snt_sim_getc - Get character from console in simulator mode
+ *
+ */
+static int
+snt_sim_getc(void)
+{
+ return readb(master_node_bedrock_address + (UART_RX << 3));
+}
+
+/**
+ * snt_sim_input_pending - Check if there is input pending in simulator mode
+ *
+ */
+static int
+snt_sim_input_pending(void)
+{
+ return readb(master_node_bedrock_address +
+ (UART_LSR << 3)) & UART_LSR_DR;
+}
+
+/* routines for an interrupt driven console (normal) */
+
+/**
+ * snt_intr_getc - Get a character from the console, interrupt mode
+ *
+ */
+static int
+snt_intr_getc(void)
+{

```

```

+ return ia64_sn_console_readc();
+}
+
+/**
+ * snt_intr_input_pending – Check if input is pending, interrupt mode
+ *
+ */
+static int
+snt_intr_input_pending(void)
+{
+ return ia64_sn_console_intr_status() & SAL_CONSOLE_INTR_RECV;
+}
+
+/* these functions are polled and interrupt */
+
+/**
+ * snt_hw_puts_raw – Send raw string to the console, polled or interrupt mode
+ * @s: String
+ * @len: Length
+ *
+ */
+static int
+snt_hw_puts_raw(const char *s, int len)
+{
+ /* this will call the PROM and not return until this is done */
+ return ia64_sn_console_putb(s, len);
+}
+
+/**
+ * snt_hw_puts_buffered – Send string to console, polled or interrupt mode
+ * @s: String
+ * @len: Length
+ *
+ */
+static int
+snt_hw_puts_buffered(const char *s, int len)
+{
+ /* queue data to the PROM */
+ return ia64_sn_console_xmit_chars((char *)s, len);
+}
+
+/* uart interface structs
+ * These functions are associated with the uart_port that the serial core
+ * infrastructure calls.
+ *
+ * Note: Due to how the console works, many routines are no-ops.
+ */
+
+/**
+ * snt_type – What type of console are we?
+ * @port: Port to operate with (we ignore since we only have one port)

```

```

+ *
+ */
+static const char *
+snp_type(struct uart_port *port)
+{
+ return ("SGI SN L1");
+}
+
+/**
+ * snp_tx_empty – Is the transmitter empty? We pretend we're always empty
+ * @port: Port to operate on (we ignore since we only have one port)
+ *
+ */
+static unsigned int
+snp_tx_empty(struct uart_port *port)
+{
+ return 1;
+}
+
+/**
+ * snp_stop_tx – stop the transmitter – no-op for us
+ * @port: Port to operate on – we ignore – no-op function
+ * @tty_stop: Set to 1 if called via uart_stop
+ *
+ */
+static void
+snp_stop_tx(struct uart_port *port, unsigned int tty_stop)
+{
+}
+
+/**
+ * snp_release_port – Free i/o and resources for port – no-op for us
+ * @port: Port to operate on – we ignore – no-op function
+ *
+ */
+static void
+snp_release_port(struct uart_port *port)
+{
+}
+
+/**
+ * snp_enable_ms – Force modem status interrupts on – no-op for us
+ * @port: Port to operate on – we ignore – no-op function
+ *
+ */
+static void
+snp_enable_ms(struct uart_port *port)
+{
+}
+
+/**

```

Linux-Kernel: [PATCH 2.6] Altix serial driver

```
+ * snp_shutdown – shut down the port – free irq and disable – no-op for us
+ * @port: Port to shut down – we ignore
+ *
+ */
+static void
+snp_shutdown(struct uart_port *port)
+{
+}
+
+/**
+ * snp_set_mctrl – set control lines (dtr, rts, etc) – no-op for our console
+ * @port: Port to operate on – we ignore
+ * @mctrl: Lines to set/unset – we ignore
+ *
+ */
+static void
+snp_set_mctrl(struct uart_port *port, unsigned int mctrl)
+{
+}
+
+/**
+ * snp_get_mctrl – get control line info, we just return a static value
+ * @port: port to operate on – we only have one port so we ignore this
+ *
+ */
+static unsigned int
+snp_get_mctrl(struct uart_port *port)
+{
+ return TIOCM_CAR | TIOCM_RNG | TIOCM_DSR | TIOCM_CTS;
+}
+
+/**
+ * snp_stop_rx – Stop the receiver – we ignore this
+ * @port: Port to operate on – we ignore
+ *
+ */
+static void
+snp_stop_rx(struct uart_port *port)
+{
+}
+
+/**
+ * snp_start_tx – Start transmitter
+ * @port: Port to operate on
+ * @tty_stop: Set to 1 if called via uart_start
+ *
+ */
+static void
+snp_start_tx(struct uart_port *port, unsigned int tty_stop)
+{
+ if (sal_console_port.sc_ops->sal_wakeup_transmit)
```

Linux-Kernel: [PATCH 2.6] Altix serial driver

```
+ sal_console_port.sc_ops->sal_wakeup_transmit(&sal_console_port, TRANSMIT_BUFFERED);
+
+}
+
+/**
+ * snp_break_ctl – handle breaks – ignored by us
+ * @port: Port to operate on
+ * @break_state: Break state
+ *
+ */
+static void
+snp_break_ctl(struct uart_port *port, int break_state)
+{
+}
+
+/**
+ * snp_startup – Start up the serial port – always return 0 (We're always on)
+ * @port: Port to operate on
+ *
+ */
+static int
+snp_startup(struct uart_port *port)
+{
+ return 0;
+}
+
+/**
+ * snp_set_termios – set termios stuff – we ignore these
+ * @port: port to operate on
+ * @termios: New settings
+ * @termios: Old
+ *
+ */
+static void
+snp_set_termios(struct uart_port *port, struct termios *termios,
+ struct termios *old)
+{
+}
+
+/**
+ * snp_request_port – allocate resources for port – ignored by us
+ * @port: port to operate on
+ *
+ */
+static int
+snp_request_port(struct uart_port *port)
+{
+ return 0;
+}
+
+/**
```

Linux-Kernel: [PATCH 2.6] Altix serial driver

```
+ * snp_config_port – allocate resources, set up – we ignore, we're always on
+ * @port: Port to operate on
+ * @flags: flags used for port setup
+ *
+ */
+static void
+snp_config_port(struct uart_port *port, int flags)
+{
+}
+
+/* Associate the uart functions above – given to serial core */
+
+static struct uart_ops sn_console_ops = {
+ .tx_empty = snp_tx_empty,
+ .set_mctrl = snp_set_mctrl,
+ .get_mctrl = snp_get_mctrl,
+ .stop_tx = snp_stop_tx,
+ .start_tx = snp_start_tx,
+ .stop_rx = snp_stop_rx,
+ .enable_ms = snp_enable_ms,
+ .break_ctl = snp_break_ctl,
+ .startup = snp_startup,
+ .shutdown = snp_shutdown,
+ .set_termios = snp_set_termios,
+ .pm = NULL,
+ .type = snp_type,
+ .release_port = snp_release_port,
+ .request_port = snp_request_port,
+ .config_port = snp_config_port,
+ .verify_port = NULL,
+};
+
+/* End of uart struct functions and defines */
+
+
+/**
+ * early_printk_sn_sal – The early printk (possible setup) and function call
+ * @s: String to send
+ * @count: length
+ *
+ */
+void
+early_printk_sn_sal(const char *s, unsigned count)
+{
+ extern void early_sn_setup(void);
+
+ if (!sal_console_port.sc_ops) {
+ if (IS_RUNNING_ON_SIMULATOR())
+ sal_console_port.sc_ops = &sim_ops;
+ else
+ sal_console_port.sc_ops = &poll_ops;
+ }
```

Linux–Kernel: [PATCH 2.6] Altix serial driver

```
+
+ early_sn_setup();
+ }
+ sal_console_port.sc_ops->sal_puts_raw(s, count);
+}
+
+/**
+ * sn_debug_printf – close to hardware debugging printf
+ * @fmt: printf format
+ *
+ * This is as "close to the metal" as we can get, used when the driver
+ * itself may be broken.
+ *
+ */
+static int
+sn_debug_printf(const char *fmt, ...)
+{
+ static char printk_buf[1024];
+ int printed_len;
+ va_list args;
+
+ va_start(args, fmt);
+ printed_len = vsnprintf(printk_buf, sizeof (printk_buf), fmt, args);
+ early_printk_sn_sal(printk_buf, printed_len);
+ va_end(args);
+ return printed_len;
+}
+
+/**
+ * Interrupt handling routines.
+ */
+
+
+/**
+ * sn_receive_chars – Grab characters, pass them to tty layer
+ * @port: Port to operate on
+ * @regs: Saved registers (needed by uart_handle_sysrq_char)
+ *
+ * Note: If we're not registered with the serial core infrastructure yet,
+ * we don't try to send characters to it...
+ *
+ */
+static void
+sn_receive_chars(struct sn_cons_port *port, struct pt_regs *regs)
+{
+ int ch;
+ struct tty_struct *tty;
+
+ if (!port) {
+ printk(KERN_ERR "sn_receive_chars – port NULL so can't receive\n");
+ return;
+ }
```

Linux–Kernel: [PATCH 2.6] Altix serial driver

```

+ }
+
+ if (!port->sc_ops) {
+ printk(KERN_ERR "sn_receive_chars - port->sc_ops NULL so can't receive\n");
+ return;
+ }
+
+ if (port->sc_port.info) {
+ /* The serial_core stuffs are initilized, use them */
+ tty = port->sc_port.info->tty;
+ }
+ else {
+ /* Not registered yet - can't pass to tty layer. */
+ tty = NULL;
+ }
+
+ while (port->sc_ops->sal_input_pending()) {
+ ch = port->sc_ops->sal_getc();
+ if (ch < 0) {
+ printk(KERN_ERR "sn_console: An error ocured while "
+ "obtaining data from the console (0x%0x)\n", ch);
+ break;
+ }
+ #if defined(CONFIG_SERIAL_SGI_L1_CONSOLE) && defined(CONFIG_MAGIC_SYSRQ)
+ if (uart_handle_sysrq_char(&port->sc_port, ch, regs))
+ continue;
+ #endif /* CONFIG_SERIAL_SGI_L1_CONSOLE && CONFIG_MAGIC_SYSRQ */
+
+ /* record the character to pass up to the tty layer */
+ if (tty) {
+ *tty->flip.char_buf_ptr = ch;
+ *tty->flip.flag_buf_ptr = TTY_NORMAL;
+ tty->flip.char_buf_ptr++;
+ tty->flip.count++;
+ if (tty->flip.count == TTY_FLIPBUF_SIZE)
+ break;
+ }
+ else {
+ }
+ port->sc_port.icount.rx++;
+ }
+
+ if (tty)
+ tty_flip_buffer_push(tty);
+ }
+
+ /**
+ * sn_transmit_chars - grab characters from serial core, send off
+ * @port: Port to operate on
+ * @raw: Transmit raw or buffered
+ *

```

Linux-Kernel: [PATCH 2.6] Altix serial driver

```
+ * Note: If we're early, before we're registered with serial core, the
+ * writes are going through sn_sal_console_write because that's how
+ * register_console has been set up. We currently could have asynch
+ * polls calling this function due to sn_sal_switch_to_asynch but we can
+ * ignore them until we register with the serial core stuffs.
+ *
+ */
+static void
+sn_transmit_chars(struct sn_cons_port *port, int raw)
+{
+ int xmit_count, tail, head, loops, ii;
+ int result;
+ char *start;
+ struct circ_buf *xmit;
+
+ if (!port)
+ return;
+
+ BUG_ON(!port->sc_is_asynch);
+
+ if (port->sc_port.info) {
+ /* We're initilized, using serial core infrastructure */
+ xmit = &port->sc_port.info->xmit;
+ }
+ else {
+ /* Probably sn_sal_switch_to_asynch has been run but serial core isn't
+ * initilized yet. Just return. Writes are going through
+ * sn_sal_console_write (due to register_console) at this time.
+ */
+ return;
+ }
+
+ if (uart_circ_empty(xmit) || uart_tx_stopped(&port->sc_port)) {
+ /* Nothing to do. */
+ return;
+ }
+
+ head = xmit->head;
+ tail = xmit->tail;
+ start = &xmit->buf[tail];
+
+ /* twice around gets the tail to the end of the buffer and
+ * then to the head, if needed */
+ loops = (head < tail) ? 2 : 1;
+
+ for (ii = 0; ii < loops; ii++) {
+ xmit_count = (head < tail) ?
+ (UART_XMIT_SIZE - tail) : (head - tail);
+
+ if (xmit_count > 0) {
+ if (raw == TRANSMIT_RAW)
```

```

+ result =
+ port->sc_ops->sal_puts_raw(start,
+ xmit_count);
+ else
+ result =
+ port->sc_ops->sal_puts(start, xmit_count);
+#ifdef DEBUG
+ if (!result)
+ sn_debug_printf("");
+#endif
+ if (result > 0) {
+ xmit_count -= result;
+ port->sc_port.icount.tx += result;
+ tail += result;
+ tail &= UART_XMIT_SIZE - 1;
+ xmit->tail = tail;
+ start = &xmit->buf[tail];
+ }
+ }
+ }
+
+ if (uart_circ_chars_pending(xmit) < WAKEUP_CHARS)
+ uart_write_wakeup(&port->sc_port);
+
+ if (uart_circ_empty(xmit))
+ snp_stop_tx(&port->sc_port, 0); /* no-op for us */
+}
+
+/**
+ * sn_sal_interrupt - Handle console interrupts
+ * @irq: irq #, useful for debug statements
+ * @dev_id: our pointer to our port (sn_cons_port which contains the uart port)
+ * @regs: Saved registers, used by sn_receive_chars for uart_handle_sysrq_char
+ *
+ */
+static irqreturn_t
+sn_sal_interrupt(int irq, void *dev_id, struct pt_regs *regs)
+{
+ struct sn_cons_port *port = (struct sn_cons_port *) dev_id;
+ unsigned long flags;
+ int status = ia64_sn_console_intr_status();
+
+ if (!port)
+ return IRQ_NONE;
+
+ spin_lock_irqsave(&port->sc_port.lock, flags);
+ if (status & SAL_CONSOLE_INTR_RECV) {
+ sn_receive_chars(port, regs);
+ }
+ if (status & SAL_CONSOLE_INTR_XMIT) {
+ sn_transmit_chars(port, TRANSMIT_BUFFERED);

```

```

+ }
+ spin_unlock_irqrestore(&port->sc_port.lock, flags);
+ return IRQ_HANDLED;
+}
+
+/**
+ * sn_sal_connect_interrupt - Request interrupt, handled by sn_sal_interrupt
+ * @port: Our sn_cons_port (which contains the uart port)
+ *
+ * returns the console irq if interrupt is successfully registered, else 0
+ *
+ */
+static int
+sn_sal_connect_interrupt(struct sn_cons_port *port)
+{
+ if (request_irq(SGI_UART_VECTOR, sn_sal_interrupt, SA_INTERRUPT,
+ "SAL console driver", port) >= 0) {
+ return SGI_UART_VECTOR;
+ }
+
+ printk(KERN_INFO "sn_console: console proceeding in polled mode\n");
+ return 0;
+}
+
+/**
+ * sn_sal_timer_poll - this function handles polled console mode
+ * @data: A pointer to our sn_cons_port (which contains the uart port)
+ *
+ * data is the pointer that init_timer will store for us. This function is
+ * associated with init_timer to see if there is any console traffic.
+ * Obviously not used in interrupt mode
+ *
+ */
+static void
+sn_sal_timer_poll(unsigned long data)
+{
+ struct sn_cons_port *port = (struct sn_cons_port *) data;
+ unsigned long flags;
+
+ if (!port)
+ return;
+
+ if (!port->sc_port.irq) {
+ spin_lock_irqsave(&port->sc_port.lock, flags);
+ sn_receive_chars(port, NULL);
+ sn_transmit_chars(port, TRANSMIT_RAW);
+ spin_unlock_irqrestore(&port->sc_port.lock, flags);
+ mod_timer(&port->sc_timer,
+ jiffies + port->sc_interrupt_timeout);
+ }
+}

```

```

+
+/*
+ * Boot-time initialization code
+ */
+
+/**
+ * sn_sal_switch_to_async - Switch to async mode (as opposed to synch)
+ * @port: Our sn_cons_port (which contains the uart port)
+ *
+ * So this is used by sn_sal_serial_console_init (early on, before we're
+ * registered with serial core). It's also used by sn_sal_module_init
+ * right after we've registered with serial core. The later only happens
+ * if we didn't already come through here via sn_sal_serial_console_init.
+ *
+ */
+static void __init
+sn_sal_switch_to_async(struct sn_cons_port *port)
+{
+ unsigned long flags;
+
+ if (!port)
+ return;
+
+ DPRINTF("sn_console: about to switch to asynchronous console\n");
+
+ /* without early_printk, we may be invoked late enough to race
+ * with other cpus doing console IO at this point, however
+ * console interrupts will never be enabled */
+ spin_lock_irqsave(&port->sc_port.lock, flags);
+
+ /* early_printk invocation may have done this for us */
+ if (!port->sc_ops) {
+ if (IS_RUNNING_ON_SIMULATOR())
+ port->sc_ops = &sim_ops;
+ else
+ port->sc_ops = &poll_ops;
+ }
+
+ /* we can't turn on the console interrupt (as request_irq
+ * calls kcalloc, which isn't set up yet), so we rely on a
+ * timer to poll for input and push data from the console
+ * buffer.
+ */
+ init_timer(&port->sc_timer);
+ port->sc_timer.function = sn_sal_timer_poll;
+ port->sc_timer.data = (unsigned long) port;
+
+ if (IS_RUNNING_ON_SIMULATOR())
+ port->sc_interrupt_timeout = 6;
+ else {
+ /* 960cps / 16 char FIFO = 60HZ

```

Linux-Kernel: [PATCH 2.6] Altix serial driver

```

+ * HZ / (SN_SAL_FIFO_SPEED_CPS / SN_SAL_FIFO_DEPTH) */
+ port->sc_interrupt_timeout =
+ HZ * SN_SAL_UART_FIFO_DEPTH / SN_SAL_UART_FIFO_SPEED_CPS;
+ }
+ mod_timer(&port->sc_timer, jiffies + port->sc_interrupt_timeout);
+
+ port->sc_is_async = 1;
+ spin_unlock_irqrestore(&port->sc_port.lock, flags);
+}
+
+/**
+ * sn_sal_switch_to_interrupts - Switch to interrupt driven mode
+ * @port: Our sn_cons_port (which contains the uart port)
+ *
+ * In sn_sal_module_init, after we're registered with serial core and
+ * the port is added, this function is called to switch us to interrupt
+ * mode. We were previously in async/polling mode (using init_timer).
+ *
+ * We attempt to switch to interrupt mode here by calling
+ * sn_sal_connect_interrupt. If that works out, we enable receive interrupts.
+ */
+static void __init
+sn_sal_switch_to_interrupts(struct sn_cons_port *port)
+{
+ int irq;
+ unsigned long flags;
+
+ if (!port)
+ return;
+
+ DPRINTF("sn_console: switching to interrupt driven console\n");
+
+ spin_lock_irqsave(&port->sc_port.lock, flags);
+
+ irq = sn_sal_connect_interrupt(port);
+
+ if (irq) {
+ port->sc_port.irq = irq;
+ port->sc_ops = &intr_ops;
+
+ /* turn on receive interrupts */
+ ia64_sn_console_intr_enable(SAL_CONSOLE_INTR_RECV);
+ }
+ spin_unlock_irqrestore(&port->sc_port.lock, flags);
+}
+
+/**
+ * Kernel console definitions
+ */
+
+#ifdef CONFIG_SERIAL_SGI_L1_CONSOLE

```

Linux-Kernel: [PATCH 2.6] Altix serial driver

```
+static void sn_sal_console_write(struct console *, const char *, unsigned);
+static int __init sn_sal_console_setup(struct console *, char *);
+extern struct uart_driver sal_console_uart;
+extern struct tty_driver *uart_console_device(struct console *, int *);
+
+static struct console sal_console = {
+ .name = DEVICE_NAME,
+ .write = sn_sal_console_write,
+ .device = uart_console_device,
+ .setup = sn_sal_console_setup,
+ .index = -1, /* unspecified */
+ .data = &sal_console_uart,
+};
+
+#define SAL_CONSOLE &sal_console
+#else
+#define SAL_CONSOLE 0
+#endif /* CONFIG_SERIAL_SGI_L1_CONSOLE */
+
+static struct uart_driver sal_console_uart = {
+ .owner = THIS_MODULE,
+ .driver_name = "sn_console",
+ .dev_name = DEVICE_NAME,
+ .major = 0, /* major/minor set at registration time per SYSFS_ONLY */
+ .minor = 0,
+ .nr = 1, /* one port */
+ .cons = SAL_CONSOLE,
+};
+
+/**
+ * sn_sal_module_init - When the kernel loads us, get us rolling w/ serial core
+ *
+ * Before this is called, we've been printing kernel messages in a special
+ * early mode not making use of the serial core infrastructure. When our
+ * driver is loaded for real, we register the driver and port with serial
+ * core and try to enable interrupt driven mode.
+ */
+static int __init
+sn_sal_module_init(void)
+{
+ int retval;
+
+
+ printk(KERN_INFO "sn_console: Console driver init\n");
+
+ if (!ia64_platform_is("sn2"))
+ return -ENODEV;
+
+ if (SYSFS_ONLY == 1) {
+ misc.minor = MISC_DYNAMIC_MINOR;
+ misc.name = DEVICE_NAME_DYNAMIC;

```

Linux-Kernel: [PATCH 2.6] Altix serial driver

```
+ retval = misc_register(&misc);
+ if (retval != 0) {
+ printk("Failed to register console device using misc_register.\n");
+ return -ENODEV;
+ }
+ sal_console_uart.major = MISC_MAJOR;
+ sa
```