

[patch 04/33] m68k: Atari keyboard and mouse support.

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2007-05/msg00624.html>

- *From:* Geert Uytterhoeven <geert@xxxxxxxxxxxxxxxx>
 - *Date:* Tue, 01 May 2007 22:32:38 +0200
-

From: Michael Schmitz <schmitz@xx>

Atari keyboard and mouse support.
(reformatting and Kconfig fixes by Roman Zippel)

Signed-off-by: Michael Schmitz <schmitz@xxxxxxxx>
Signed-off-by: Roman Zippel <zippel@xxxxxxxxxxxxxxxx>
Signed-off-by: Geert Uytterhoeven <geert@xxxxxxxxxxxxxxxx>

arch/m68k/Kconfig | 3
arch/m68k/atari/Makefile | 1
arch/m68k/atari/atakeyb.c | 730 +++
drivers/input/keyboard/Kconfig | 11
drivers/input/keyboard/Makefile | 1
drivers/input/keyboard/atakbd.c | 134 +++++++
drivers/input/mouse/Kconfig | 11
drivers/input/mouse/Makefile | 1
drivers/input/mouse/atarimouse.c | 160 +++++++
include/asm-m68k/atarikb.h | 6
include/linux/input.h | 1
11 files changed, 1059 insertions(+)

--- linux-m68k-2.6.21.orig/arch/m68k/Kconfig
+++ linux-m68k-2.6.21/arch/m68k/Kconfig
@@ -409,6 +409,9 @@ config STRAM_PROC
help
Say Y here to report ST-RAM usage statistics in /proc/stram.

+config ATARI_KBD_CORE
+ bool
+
config HEARTBEAT
bool "Use power LED as a heartbeat" if AMIGA || APOLLO || ATARI || MAC || Q40
default y if !AMIGA && !APOLLO && !ATARI && !MAC && !Q40 && HP300
--- linux-m68k-2.6.21.orig/arch/m68k/atari/Makefile
+++ linux-m68k-2.6.21/arch/m68k/atari/Makefile
@@ -8,3 +8,4 @@ obj-y := config.o time.o debug.o ataint
ifeq (\$(CONFIG_PCI),y)

[patch 04/33] m68k: Atari keyboard and mouse support.

```
obj-$(CONFIG_HADES) += hades-pci.o
endif
+obj-$(CONFIG_ATARI_KBD_CORE) += atakeyb.o
--- /dev/null
+++ linux-m68k-2.6.21/arch/m68k/atari/atakeyb.c
@@ -0,0 +1,730 @@
+/*
+ * linux/atari/atakeyb.c
+ *
+ * Atari Keyboard driver for 680x0 Linux
+ *
+ * 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.
+ */
+
+/* Atari support by Robert de Vries
+ * enhanced by Bjoern Brauel and Roman Hodek
+ */
+
+#include <linux/sched.h>
+#include <linux/kernel.h>
+#include <linux/interrupt.h>
+#include <linux/errno.h>
+#include <linux/keyboard.h>
+#include <linux/delay.h>
+#include <linux/timer.h>
+#include <linux/kd.h>
+#include <linux/random.h>
+#include <linux/init.h>
+#include <linux/kbd_kern.h>
+
+#include <asm/atariints.h>
+#include <asm/atarihw.h>
+#include <asm/atarikb.h>
+#include <asm/atari_joystick.h>
+#include <asm/irq.h>
+
+static void atakeyb_rep(unsigned long ignore);
+extern unsigned int keymap_count;
+
+/* Hook for MIDI serial driver */
+void (*atari_MIDI_interrupt_hook) (void);
+/* Hook for mouse driver */
+void (*atari_mouse_interrupt_hook) (char *);
+/* Hook for keyboard inputdev driver */
+void (*atari_input_keyboard_interrupt_hook) (unsigned char, char);
+/* Hook for mouse inputdev driver */
+void (*atari_input_mouse_interrupt_hook) (char *);
+
```

[patch 04/33] m68k: Atari keyboard and mouse support.

```
+/* variables for IKBD self test: */
+
+/* state: 0: off; >0: in progress; >1: 0xf1 received */
+static volatile int ikbd_self_test;
+/* timestamp when last received a char */
+static volatile unsigned long self_test_last_rcv;
+/* bitmap of keys reported as broken */
+static unsigned long broken_keys[128/(sizeof(unsigned long)*8)] = { 0, };
+
+#define BREAK_MASK (0x80)
+
+/*
+ * ++roman: The following changes were applied manually:
+ *
+ * - The Alt (= Meta) key works in combination with Shift and
+ * Control, e.g. Alt+Shift+a sends Meta-A (0xc1), Alt+Control+A sends
+ * Meta-Ctrl-A (0x81) ...
+ *
+ * - The parentheses on the keypad send '(' and ')' with all
+ * modifiers (as would do e.g. keypad '+'), but they cannot be used as
+ * application keys (i.e. sending Esc O c).
+ *
+ * - HELP and UNDO are mapped to be F21 and F24, resp, that send the
+ * codes "\E[M" and "\E[P". (This is better than the old mapping to
+ * F11 and F12, because these codes are on Shift+F1/2 anyway.) This
+ * way, applications that allow their own keyboard mappings
+ * (e.g. tcsh, X Windows) can be configured to use them in the way
+ * the label suggests (providing help or undoing).
+ *
+ * - Console switching is done with Alt+Fx (consoles 1..10) and
+ * Shift+Alt+Fx (consoles 11..20).
+ *
+ * - The misc. special function implemented in the kernel are mapped
+ * to the following key combinations:
+ *
+ * ClrHome -> Home/Find
+ * Shift + ClrHome -> End/Select
+ * Shift + Up -> Page Up
+ * Shift + Down -> Page Down
+ * Alt + Help -> show system status
+ * Shift + Help -> show memory info
+ * Ctrl + Help -> show registers
+ * Ctrl + Alt + Del -> Reboot
+ * Alt + Undo -> switch to last console
+ * Shift + Undo -> send interrupt
+ * Alt + Insert -> stop/start output (same as ^S/^Q)
+ * Alt + Up -> Scroll back console (if implemented)
+ * Alt + Down -> Scroll forward console (if implemented)
+ * Alt + CapsLock -> NumLock
+ *
+ * ++Andreas:
```

[patch 04/33] m68k: Atari keyboard and mouse support.

```
+ *
+ * - Help mapped to K_HELP
+ * - Undo mapped to K_UNDO (= K_F246)
+ * - Keypad Left/Right Parenthesis mapped to new K_PPAREN[LR]
+ */
+
+static u_short ataplain_map[NR_KEYS] __initdata = {
+ 0xf200, 0xf01b, 0xf031, 0xf032, 0xf033, 0xf034, 0xf035, 0xf036,
+ 0xf037, 0xf038, 0xf039, 0xf030, 0xf02d, 0xf03d, 0xf008, 0xf009,
+ 0xfb71, 0xfb77, 0xfb65, 0xfb72, 0xfb74, 0xfb79, 0xfb75, 0xfb69,
+ 0xfb6f, 0xfb70, 0xf05b, 0xf05d, 0xf201, 0xf702, 0xfb61, 0xfb73,
+ 0xfb64, 0xfb66, 0xfb67, 0xfb68, 0xfb6a, 0xfb6b, 0xfb6c, 0xf03b,
+ 0xf027, 0xf060, 0xf700, 0xf05c, 0xfb7a, 0xfb78, 0xfb63, 0xfb76,
+ 0xfb62, 0xfb6e, 0xfb6d, 0xf02c, 0xf02e, 0xf02f, 0xf700, 0xf200,
+ 0xf703, 0xf020, 0xf207, 0xf100, 0xf101, 0xf102, 0xf103, 0xf104,
+ 0xf105, 0xf106, 0xf107, 0xf108, 0xf109, 0xf200, 0xf200, 0xf114,
+ 0xf603, 0xf200, 0xf30b, 0xf601, 0xf200, 0xf602, 0xf30a, 0xf200,
+ 0xf600, 0xf200, 0xf115, 0xf07f, 0xf200, 0xf200, 0xf200, 0xf200,
+ 0xf200, 0xf200, 0xf200, 0xf200, 0xf200, 0xf200, 0xf200, 0xf200,
+ 0xf200, 0xf1ff, 0xf11b, 0xf312, 0xf313, 0xf30d, 0xf30c, 0xf307,
+ 0xf308, 0xf309, 0xf304, 0xf305, 0xf306, 0xf301, 0xf302, 0xf303,
+ 0xf300, 0xf310, 0xf30e, 0xf200, 0xf200, 0xf200, 0xf200, 0xf200,
+ 0xf200, 0xf200, 0xf200, 0xf200, 0xf200, 0xf200, 0xf200, 0xf200
+};
+
+typedef enum kb_state_t {
+ KEYBOARD, AMOUSE, RMOUSE, JOYSTICK, CLOCK, RESYNC
+} KB_STATE_T;
+
+#define IS_SYNC_CODE(sc) ((sc) >= 0x04 && (sc) <= 0xfb)
+
+typedef struct keyboard_state {
+ unsigned char buf[6];
+ int len;
+ KB_STATE_T state;
+} KEYBOARD_STATE;
+
+KEYBOARD_STATE kb_state;
+
+#define DEFAULT_KEYB_REP_DELAY (HZ/4)
+#define DEFAULT_KEYB_REP_RATE (HZ/25)
+
+/* These could be settable by some ioctl() in future... */
+static unsigned int key_repeat_delay = DEFAULT_KEYB_REP_DELAY;
+static unsigned int key_repeat_rate = DEFAULT_KEYB_REP_RATE;
+
+static unsigned char rep_scancode;
+static struct timer_list atakeyb_rep_timer = {
+ .function = atakeyb_rep,
+};
+
```

[patch 04/33] m68k: Atari keyboard and mouse support.

```
+static void atakeyb_rep(unsigned long ignore)
+{
+ /* Disable keyboard for the time we call handle_scancode(), else a race
+ * in the keyboard tty queue may happen */
+ atari_disable_irq(IRQ_MFP_ACIA);
+ del_timer(&atakeyb_rep_timer);
+
+ /* A keyboard int may have come in before we disabled the irq, so
+ * double-check whether rep_scancode is still != 0 */
+ if (rep_scancode) {
+ init_timer(&atakeyb_rep_timer);
+ atakeyb_rep_timer.expires = jiffies + key_repeat_rate;
+ add_timer(&atakeyb_rep_timer);
+
+ //handle_scancode(rep_scancode, 1);
+ if (atari_input_keyboard_interrupt_hook)
+ atari_input_keyboard_interrupt_hook(rep_scancode, 1);
+ }
+
+ atari_enable_irq(IRQ_MFP_ACIA);
+}
+
+
+ /* ++roman: If a keyboard overrun happened, we can't tell in general how much
+ * bytes have been lost and in which state of the packet structure we are now.
+ * This usually causes keyboards bytes to be interpreted as mouse movements
+ * and vice versa, which is very annoying. It seems better to throw away some
+ * bytes (that are usually mouse bytes) than to misinterpret them. Therefor I
+ * introduced the RESYNC state for IKBD data. In this state, the bytes up to
+ * one that really looks like a key event (0x04..0xf2) or the start of a mouse
+ * packet (0xf8..0xfb) are thrown away, but at most 2 bytes. This at least
+ * speeds up the resynchronization of the event structure, even if maybe a
+ * mouse movement is lost. However, nothing is perfect. For bytes 0x01..0x03,
+ * it's really hard to decide whether they're mouse or keyboard bytes. Since
+ * overruns usually occur when moving the Atari mouse rapidly, they're seen as
+ * mouse bytes here. If this is wrong, only a make code of the keyboard gets
+ * lost, which isn't too bad. Loosing a break code would be disastrous,
+ * because then the keyboard repeat strikes...
+ */
+
+static irqreturn_t atari_keyboard_interrupt(int irq, void *dummy)
+{
+ u_char acia_stat;
+ int scancode;
+ int break_flag;
+
+repeat:
+ if (acia.mid_ctrl & ACIA_IRQ)
+ if (atari_MIDI_interrupt_hook)
+ atari_MIDI_interrupt_hook();
+ acia_stat = acia.key_ctrl;
```

[patch 04/33] m68k: Atari keyboard and mouse support.

```
+ /* check out if the interrupt came from this ACIA */
+ if (!(acia_stat | acia.mid_ctrl) & ACIA_IRQ))
+ return IRQ_HANDLED;
+
+ if (acia_stat & ACIA_OVRN) {
+ /* a very fast typist or a slow system, give a warning */
+ /* ...happens often if interrupts were disabled for too long */
+ printk(KERN_DEBUG "Keyboard overrun\n");
+ scancode = acia.key_data;
+ /* Turn off autorepeating in case a break code has been lost */
+ del_timer(&atakeyb_rep_timer);
+ rep_scancode = 0;
+ if (ikbd_self_test)
+ /* During self test, don't do resyncing, just process the code */
+ goto interpret_scancode;
+ else if (IS_SYNC_CODE(scancode)) {
+ /* This code seem already to be the start of a new packet or a
+ * single scancode */
+ kb_state.state = KEYBOARD;
+ goto interpret_scancode;
+ } else {
+ /* Go to RESYNC state and skip this byte */
+ kb_state.state = RESYNC;
+ kb_state.len = 1; /* skip max. 1 another byte */
+ goto repeat;
+ }
+ }
+
+ if (acia_stat & ACIA_RDRF) {
+ /* received a character */
+ scancode = acia.key_data; /* get it or reset the ACIA, I'll get it! */
+ tasklet_schedule(&keyboard_tasklet);
+ interpret_scancode:
+ switch (kb_state.state) {
+ case KEYBOARD:
+ switch (scancode) {
+ case 0xF7:
+ kb_state.state = AMOUSE;
+ kb_state.len = 0;
+ break;
+
+ case 0xF8:
+ case 0xF9:
+ case 0xFA:
+ case 0xFB:
+ kb_state.state = RMOUSE;
+ kb_state.len = 1;
+ kb_state.buf[0] = scancode;
+ break;
+
+ case 0xFC:
```

[patch 04/33] m68k: Atari keyboard and mouse support.

```
+ kb_state.state = CLOCK;
+ kb_state.len = 0;
+ break;
+
+ case 0xFE:
+ case 0xFF:
+ kb_state.state = JOYSTICK;
+ kb_state.len = 1;
+ kb_state.buf[0] = scancode;
+ break;
+
+ case 0xF1:
+ /* during self-test, note that 0xf1 received */
+ if (ikbd_self_test) {
+ ++ikbd_self_test;
+ self_test_last_rcv = jiffies;
+ break;
+ }
+ /* FALL THROUGH */
+
+ default:
+ break_flag = scancode & BREAK_MASK;
+ scancode &= ~BREAK_MASK;
+ if (ikbd_self_test) {
+ /* Scancodes sent during the self-test stand for broken
+ * keys (keys being down). The code *should* be a break
+ * code, but nevertheless some AT keyboard interfaces send
+ * make codes instead. Therefore, simply ignore
+ * break_flag...
+ */
+ int keyval = plain_map[scancode], keytyp;
+
+ set_bit(scancode, broken_keys);
+ self_test_last_rcv = jiffies;
+ keyval = plain_map[scancode];
+ keytyp = KTYP(keyval) - 0xf0;
+ keyval = KVAL(keyval);
+
+ printk(KERN_WARNING "Key with scancode %d ", scancode);
+ if (keytyp == KT_LATIN || keytyp == KT_LETTER) {
+ if (keyval < ' ')
+ printk("(^%c) ", keyval + '@');
+ else
+ printk("(%c) ", keyval);
+ }
+ printk("is broken -- will be ignored.\n");
+ break;
+ } else if (test_bit(scancode, broken_keys))
+ break;
+
+ #if 0 // FIXME; hangs at boot
```

[patch 04/33] m68k: Atari keyboard and mouse support.

```
+ if (break_flag) {
+ del_timer(&atakeyb_rep_timer);
+ rep_scancode = 0;
+ } else {
+ del_timer(&atakeyb_rep_timer);
+ rep_scancode = scancode;
+ atakeyb_rep_timer.expires = jiffies + key_repeat_delay;
+ add_timer(&atakeyb_rep_timer);
+ }
+ #endif
+
+ // handle_scancode(scancode, !break_flag);
+ if (atari_input_keyboard_interrupt_hook)
+ atari_input_keyboard_interrupt_hook((unsigned char)scancode, !break_flag);
+ break;
+ }
+ break;
+
+ case AMOUSE:
+ kb_state.buf[kb_state.len++] = scancode;
+ if (kb_state.len == 5) {
+ kb_state.state = KEYBOARD;
+ /* not yet used */
+ /* wake up someone waiting for this */
+ }
+ break;
+
+ case RMOUSE:
+ kb_state.buf[kb_state.len++] = scancode;
+ if (kb_state.len == 3) {
+ kb_state.state = KEYBOARD;
+ if (atari_mouse_interrupt_hook)
+ atari_mouse_interrupt_hook(kb_state.buf);
+ }
+ break;
+
+ case JOYSTICK:
+ kb_state.buf[1] = scancode;
+ kb_state.state = KEYBOARD;
+ #ifdef FIXED_ATARI_JOYSTICK
+ atari_joystick_interrupt(kb_state.buf);
+ #endif
+ break;
+
+ case CLOCK:
+ kb_state.buf[kb_state.len++] = scancode;
+ if (kb_state.len == 6) {
+ kb_state.state = KEYBOARD;
+ /* wake up someone waiting for this.
+ But will this ever be used, as Linux keeps its own time.
+ Perhaps for synchronization purposes? */
```

[patch 04/33] m68k: Atari keyboard and mouse support.

```
+ /* wake_up_interruptible(&clock_wait); */
+ }
+ break;
+
+ case RESYNC:
+ if (kb_state.len <= 0 || IS_SYNC_CODE(scancode)) {
+ kb_state.state = KEYBOARD;
+ goto interpret_scancode;
+ }
+ kb_state.len--;
+ break;
+ }
+ }
+
+ #if 0
+ if (acia_stat & ACIA_CTS)
+ /* cannot happen */;
+ #endif
+
+ if (acia_stat & (ACIA_FE | ACIA_PE)) {
+ printk("Error in keyboard communication\n");
+ }
+
+ /* handle_scancode() can take a lot of time, so check again if
+ * some character arrived
+ */
+ goto repeat;
+ }
+
+ /*
+ * I write to the keyboard without using interrupts, I poll instead.
+ * This takes for the maximum length string allowed (7) at 7812.5 baud
+ * 8 data 1 start 1 stop bit: 9.0 ms
+ * If this takes too long for normal operation, interrupt driven writing
+ * is the solution. (I made a feeble attempt in that direction but I
+ * kept it simple for now.)
+ */
+ void ikbd_write(const char *str, int len)
+ {
+ u_char acia_stat;
+
+ if ((len < 1) || (len > 7))
+ panic("ikbd: maximum string length exceeded");
+ while (len) {
+ acia_stat = acia.key_ctrl;
+ if (acia_stat & ACIA_TDRE) {
+ acia.key_data = *str++;
+ len--;
+ }
+ }
+ }
```

[patch 04/33] m68k: Atari keyboard and mouse support.

```
+
+/* Reset (without touching the clock) */
+void ikbd_reset(void)
+{
+ static const char cmd[2] = { 0x80, 0x01 };
+
+ ikbd_write(cmd, 2);
+
+ /*
+ * if all's well code 0xF1 is returned, else the break codes of
+ * all keys making contact
+ */
+}
+
+/* Set mouse button action */
+void ikbd_mouse_button_action(int mode)
+{
+ char cmd[2] = { 0x07, mode };
+
+ ikbd_write(cmd, 2);
+}
+
+/* Set relative mouse position reporting */
+void ikbd_mouse_rel_pos(void)
+{
+ static const char cmd[1] = { 0x08 };
+
+ ikbd_write(cmd, 1);
+}
+
+/* Set absolute mouse position reporting */
+void ikbd_mouse_abs_pos(int xmax, int ymax)
+{
+ char cmd[5] = { 0x09, xmax>>8, xmax&0xFF, ymax>>8, ymax&0xFF };
+
+ ikbd_write(cmd, 5);
+}
+
+/* Set mouse keycode mode */
+void ikbd_mouse_kbd_mode(int dx, int dy)
+{
+ char cmd[3] = { 0x0A, dx, dy };
+
+ ikbd_write(cmd, 3);
+}
+
+/* Set mouse threshold */
+void ikbd_mouse_thresh(int x, int y)
+{
+ char cmd[3] = { 0x0B, x, y };
+
+
```

[patch 04/33] m68k: Atari keyboard and mouse support.

```
+ ikbd_write(cmd, 3);
+}
+
+/* Set mouse scale */
+void ikbd_mouse_scale(int x, int y)
+{
+ char cmd[3] = { 0x0C, x, y };
+
+ ikbd_write(cmd, 3);
+}
+
+/* Interrogate mouse position */
+void ikbd_mouse_pos_get(int *x, int *y)
+{
+ static const char cmd[1] = { 0x0D };
+
+ ikbd_write(cmd, 1);
+
+ /* wait for returning bytes */
+}
+
+/* Load mouse position */
+void ikbd_mouse_pos_set(int x, int y)
+{
+ char cmd[6] = { 0x0E, 0x00, x>>8, x&0xFF, y>>8, y&0xFF };
+
+ ikbd_write(cmd, 6);
+}
+
+/* Set Y=0 at bottom */
+void ikbd_mouse_y0_bot(void)
+{
+ static const char cmd[1] = { 0x0F };
+
+ ikbd_write(cmd, 1);
+}
+
+/* Set Y=0 at top */
+void ikbd_mouse_y0_top(void)
+{
+ static const char cmd[1] = { 0x10 };
+
+ ikbd_write(cmd, 1);
+}
+
+/* Resume */
+void ikbd_resume(void)
+{
+ static const char cmd[1] = { 0x11 };
+
+ ikbd_write(cmd, 1);
```

```
+}
+
+/* Disable mouse */
+void ikbd_mouse_disable(void)
+{
+ static const char cmd[1] = { 0x12 };
+
+ ikbd_write(cmd, 1);
+}
+
+/* Pause output */
+void ikbd_pause(void)
+{
+ static const char cmd[1] = { 0x13 };
+
+ ikbd_write(cmd, 1);
+}
+
+/* Set joystick event reporting */
+void ikbd_joystick_event_on(void)
+{
+ static const char cmd[1] = { 0x14 };
+
+ ikbd_write(cmd, 1);
+}
+
+/* Set joystick interrogation mode */
+void ikbd_joystick_event_off(void)
+{
+ static const char cmd[1] = { 0x15 };
+
+ ikbd_write(cmd, 1);
+}
+
+/* Joystick interrogation */
+void ikbd_joystick_get_state(void)
+{
+ static const char cmd[1] = { 0x16 };
+
+ ikbd_write(cmd, 1);
+}
+
+#if 0
+/* This disables all other ikbd activities !!!! */
+/* Set joystick monitoring */
+void ikbd_joystick_monitor(int rate)
+{
+ static const char cmd[2] = { 0x17, rate };
+
+ ikbd_write(cmd, 2);
+}
+
```

[patch 04/33] m68k: Atari keyboard and mouse support.

```
+ kb_state.state = JOYSTICK_MONITOR;
+}
+ #endif
+
+ /* some joystick routines not in yet (0x18-0x19) */
+
+ /* Disable joysticks */
+ void ikbd_joystick_disable(void)
+ {
+     static const char cmd[1] = { 0x1A };
+
+     ikbd_write(cmd, 1);
+ }
+
+ /* Time-of-day clock set */
+ void ikbd_clock_set(int year, int month, int day, int hour, int minute, int second)
+ {
+     char cmd[7] = { 0x1B, year, month, day, hour, minute, second };
+
+     ikbd_write(cmd, 7);
+ }
+
+ /* Interrogate time-of-day clock */
+ void ikbd_clock_get(int *year, int *month, int *day, int *hour, int *minute, int second)
+ {
+     static const char cmd[1] = { 0x1C };
+
+     ikbd_write(cmd, 1);
+ }
+
+ /* Memory load */
+ void ikbd_mem_write(int address, int size, char *data)
+ {
+     panic("Attempt to write data into keyboard memory");
+ }
+
+ /* Memory read */
+ void ikbd_mem_read(int address, char data[6])
+ {
+     char cmd[3] = { 0x21, address>>8, address&0xFF };
+
+     ikbd_write(cmd, 3);
+
+     /* receive data and put it in data */
+ }
+
+ /* Controller execute */
+ void ikbd_exec(int address)
+ {
+     char cmd[3] = { 0x22, address>>8, address&0xFF };
+
+ }
```

[patch 04/33] m68k: Atari keyboard and mouse support.

```
+ ikbd_write(cmd, 3);
+}
+
+/* Status inquiries (0x87-0x9A) not yet implemented */
+
+/* Set the state of the caps lock led. */
+void atari_kbd_leds(unsigned int leds)
+{
+ char cmd[6] = {32, 0, 4, 1, 254 + ((leds & 4) != 0), 0};
+
+ ikbd_write(cmd, 6);
+}
+
+/*
+ * The original code sometimes left the interrupt line of
+ * the ACIAs low forever. I hope, it is fixed now.
+ *
+ * Martin Rogge, 20 Aug 1995
+ */
+
+static int atari_keyb_done = 0;
+
+int __init atari_keyb_init(void)
+{
+ if (atari_keyb_done)
+ return 0;
+
+ /* setup key map */
+ memcpy(key_maps[0], ataplain_map, sizeof(plain_map));
+
+ kb_state.state = KEYBOARD;
+ kb_state.len = 0;
+
+ request_irq(IRQ_MFP_ACIA, atari_keyboard_interrupt, IRQ_TYPE_SLOW,
+ "keyboard/mouse/MIDI", atari_keyboard_interrupt);
+
+ atari_turnoff_irq(IRQ_MFP_ACIA);
+ do {
+ /* reset IKBD ACIA */
+ acia.key_ctrl = ACIA_RESET |
+ (atari_switches & ATARI_SWITCH_IKBD) ? ACIA_RHTID : 0;
+ (void)acia.key_ctrl;
+ (void)acia.key_data;
+
+ /* reset MIDI ACIA */
+ acia.mid_ctrl = ACIA_RESET |
+ (atari_switches & ATARI_SWITCH_MIDI) ? ACIA_RHTID : 0;
+ (void)acia.mid_ctrl;
+ (void)acia.mid_data;
+
+ /* divide 500kHz by 64 gives 7812.5 baud */
```

[patch 04/33] m68k: Atari keyboard and mouse support.

```
+ /* 8 data no parity 1 start 1 stop bit */
+ /* receive interrupt enabled */
+ /* RTS low (except if switch selected), transmit interrupt disabled */
+ acia.key_ctrl = (ACIA_DIV64|ACIA_D8N1S|ACIA_RIE) |
+ ((atari_switches & ATARI_SWITCH_IKBD) ?
+ ACIA_RHTID : ACIA_RLTID);
+
+ acia.mid_ctrl = ACIA_DIV16 | ACIA_D8N1S |
+ (atari_switches & ATARI_SWITCH_MIDI) ? ACIA_RHTID : 0;
+
+ /* make sure the interrupt line is up */
+ } while ((mfp.par_dt_reg & 0x10) == 0);
+
+ /* enable ACIA Interrupts */
+ mfp.active_edge &= ~0x10;
+ atari_turnon_irq(IRQ_MFP_ACIA);
+
+ ikbd_self_test = 1;
+ ikbd_reset();
+ /* wait for a period of inactivity (here: 0.25s), then assume the IKBD's
+ * self-test is finished */
+ self_test_last_rcv = jiffies;
+ while (time_before(jiffies, self_test_last_rcv + HZ/4))
+ barrier();
+ /* if not incremented: no 0xf1 received */
+ if (ikbd_self_test == 1)
+ printk(KERN_ERR "WARNING: keyboard self test failed!\n");
+ ikbd_self_test = 0;
+
+ ikbd_mouse_disable();
+ ikbd_joystick_disable();
+
+#ifdef FIXED_ATARI_JOYSTICK
+ atari_joystick_init();
+#endif
+
+ // flag init done
+ atari_keyb_done = 1;
+ return 0;
+}
+
+
+int atari_kbdrate(struct kbd_repeat *k)
+{
+ if (k->delay > 0) {
+ /* convert from msec to jiffies */
+ key_repeat_delay = (k->delay * HZ + 500) / 1000;
+ if (key_repeat_delay < 1)
+ key_repeat_delay = 1;
+ }
+ if (k->period > 0) {
```

[patch 04/33] m68k: Atari keyboard and mouse support.

```
+ key_repeat_rate = (k->period * HZ + 500) / 1000;
+ if (key_repeat_rate < 1)
+ key_repeat_rate = 1;
+ }
+
+ k->delay = key_repeat_delay * 1000 / HZ;
+ k->period = key_repeat_rate * 1000 / HZ;
+
+ return 0;
+}
+
+int atari_kbd_translate(unsigned char keycode, unsigned char *keycodep, char raw_mode)
+{
+#ifdef CONFIG_MAGIC_SYSRQ
+ /* ALT+HELP pressed? */
+ if ((keycode == 98) && ((shift_state & 0xff) == 8))
+ *keycodep = 0xff;
+ else
+#endif
+ *keycodep = keycode;
+ return 1;
+}
```

--- linux-m68k-2.6.21.orig/drivers/input/keyboard/Kconfig

+++ linux-m68k-2.6.21/drivers/input/keyboard/Kconfig

@@ -164,6 +164,17 @@ config KEYBOARD_AMIGA

To compile this driver as a module, choose M here: the module will be called amikbd.

```
+config KEYBOARD_ATARI
```

```
+ tristate "Atari keyboard"
```

```
+ depends on ATARI
```

```
+ select ATARI_KBD_CORE
```

```
+ help
```

```
+ Say Y here if you are running Linux on any Atari and have a keyboard attached.
```

```
+
```

```
+ To compile this driver as a module, choose M here: the
```

```
+ module will be called atakbd.
```

```
+
```

```
config KEYBOARD_HIL_OLD
```

```
tristate "HP HIL keyboard support (simple driver)"
```

```
depends on GSC || HP300
```

--- linux-m68k-2.6.21.orig/drivers/input/keyboard/Makefile

+++ linux-m68k-2.6.21/drivers/input/keyboard/Makefile

@@ -9,6 +9,7 @@ obj-\$(CONFIG_KEYBOARD_SUNKBD) += sunkbd

obj-\$(CONFIG_KEYBOARD_LKKBD) += lkkbd.o

obj-\$(CONFIG_KEYBOARD_XTKBD) += xtkbd.o

obj-\$(CONFIG_KEYBOARD_AMIGA) += amikbd.o

+obj-\$(CONFIG_KEYBOARD_ATARI) += atakbd.o

obj-\$(CONFIG_KEYBOARD_LOCOMO) += locomokbd.o

obj-\$(CONFIG_KEYBOARD_NEWTON) += newtonkbd.o

[patch 04/33] m68k: Atari keyboard and mouse support.

```
obj-$(CONFIG_KEYBOARD_STOWAWAY) += stowaway.o
--- /dev/null
+++ linux-m68k-2.6.21/drivers/input/keyboard/atakbd.c
@@ -0,0 +1,134 @@
+/*
+ * atakbd.c
+ *
+ * Copyright (c) 2005 Michael Schmitz
+ *
+ * Based on amikbd.c, which is
+ *
+ * Copyright (c) 2000-2001 Vojtech Pavlik
+ *
+ * Based on the work of:
+ * Hamish Macdonald
+ */
+
+
+/* Atari keyboard driver for Linux/m68k
+ *
+ * The low level init and interrupt stuff is handled in arch/mm68k/atari/atakeyb.c
+ * (the keyboard ACIA also handles the mouse and joystick data, and the keyboard
+ * interrupt is shared with the MIDI ACIA so MIDI data also get handled there).
+ * This driver only deals with handing key events off to the input layer.
+ */
+
+
+/* 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
+ *
+ * Should you need to contact me, the author, you can do so either by
+ * e-mail - mail your message to <vojtech@xxxxxx>, or by paper mail:
+ * Vojtech Pavlik, Simunkova 1594, Prague 8, 182 00 Czech Republic
+ */
+
+#include <linux/module.h>
+#include <linux/init.h>
+#include <linux/input.h>
+#include <linux/delay.h>
+#include <linux/interrupt.h>
```

[patch 04/33] m68k: Atari keyboard and mouse support.

```
+
+#include <asm/atariints.h>
+#include <asm/atarihw.h>
+#include <asm/atarikb.h>
+#include <asm/irq.h>
+
+MODULE_AUTHOR("Michael Schmitz <schmitz@xxxxxxxxxxxxxxxxxxxxxxxxxxxx>");
+MODULE_DESCRIPTION("Atari keyboard driver");
+MODULE_LICENSE("GPL");
+
+static unsigned char atakbd_keycode[0x72];
+
+static struct input_dev *atakbd_dev;
+
+static void atakbd_interrupt(unsigned char scancode, char down)
+{
+
+ if (scancode < 0x72) { /* scancodes < 0xf2 are keys */
+
+ // report raw events here?
+
+ scancode = atakbd_keycode[scancode];
+
+ if (scancode == KEY_CAPSLOCK) { /* CapsLock is a toggle switch key on Amiga */
+ input_report_key(atakbd_dev, scancode, 1);
+ input_report_key(atakbd_dev, scancode, 0);
+ input_sync(atakbd_dev);
+ } else {
+ input_report_key(atakbd_dev, scancode, down);
+ input_sync(atakbd_dev);
+ }
+ } else /* scancodes >= 0xf2 are mouse data, most likely */
+ printk(KERN_INFO "atakbd: unhandled scancode %x\n", scancode);
+
+ return;
+ }
+
+static int __init atakbd_init(void)
+{
+ int i;
+
+ if (!ATARIHW_PRESENT(ST_MFP))
+ return -EIO;
+
+ // TODO: request_mem_region if not done in arch code
+
+ if (!(atakbd_dev = input_allocate_device()))
+ return -ENOMEM;
+
+ // need to init core driver if not already done so
+ if (atari_keyb_init())
```

[patch 04/33] m68k: Atari keyboard and mouse support.

```
+ return -ENODEV;
+
+ atakbd_dev->name = "Atari Keyboard";
+ atakbd_dev->phys = "atakbd/input0";
+ atakbd_dev->id.bustype = BUS_ATARI;
+ atakbd_dev->id.vendor = 0x0001;
+ atakbd_dev->id.product = 0x0001;
+ atakbd_dev->id.version = 0x0100;
+
+ atakbd_dev->evbit[0] = BIT(EV_KEY) | BIT(EV_REP);
+ atakbd_dev->keycode = atakbd_keycode;
+ atakbd_dev->keycodesize = sizeof(unsigned char);
+ atakbd_dev->keycodemax = ARRAY_SIZE(atakbd_keycode);
+
+ for (i = 1; i < 0x72; i++) {
+ atakbd_keycode[i] = i;
+ set_bit(atakbd_keycode[i], atakbd_dev->keybit);
+ }
+
+ input_register_device(atakbd_dev);
+
+ atari_input_keyboard_interrupt_hook = atakbd_interrupt;
+
+ printk(KERN_INFO "input: %s at IKBD ACIA\n", atakbd_dev->name);
+
+ return 0;
+}
+
+static void __exit atakbd_exit(void)
+{
+ atari_input_keyboard_interrupt_hook = NULL;
+ input_unregister_device(atakbd_dev);
+}
+
+module_init(atakbd_init);
+module_exit(atakbd_exit);
--- linux-m68k-2.6.21.orig/drivers/input/mouse/Kconfig
+++ linux-m68k-2.6.21/drivers/input/mouse/Kconfig
@@ -96,6 +96,17 @@ config MOUSE_AMIGA
To compile this driver as a module, choose M here: the
module will be called amimouse.

+config MOUSE_ATARI
+ tristate "Atari mouse"
+ depends on ATARI
+ select ATARI_KBD_CORE
+ help
+ Say Y here if you have an Atari and want its native mouse
+ supported by the kernel.
+
+ To compile this driver as a module, choose M here: the
```

[patch 04/33] m68k: Atari keyboard and mouse support.

```
+ module will be called atarimouse.
+
config MOUSE_RISCPC
tristate "Acorn RiscPC mouse"
depends on ARCH_ACORN
--- linux-m68k-2.6.21.orig/drivers/input/mouse/Makefile
+++ linux-m68k-2.6.21/drivers/input/mouse/Makefile
@@ -5,6 +5,7 @@
# Each configuration option enables a list of files.

obj-$(CONFIG_MOUSE_AMIGA) += amimouse.o
+obj-$(CONFIG_MOUSE_ATARI) += atarimouse.o
obj-$(CONFIG_MOUSE_RISCPC) += rpcmouse.o
obj-$(CONFIG_MOUSE_INPORT) += inport.o
obj-$(CONFIG_MOUSE_LOGIBM) += logibm.o
--- /dev/null
+++ linux-m68k-2.6.21/drivers/input/mouse/atarimouse.c
@@ -0,0 +1,160 @@
+/*
+ * Atari mouse driver for Linux/m68k
+ *
+ * Copyright (c) 2005 Michael Schmitz
+ *
+ * Based on:
+ * Amiga mouse driver for Linux/m68k
+ *
+ * Copyright (c) 2000-2002 Vojtech Pavlik
+ *
+ */
+
+ * The low level init and interrupt stuff is handled in arch/mm68k/atari/atakeyb.c
+ * (the keyboard ACIA also handles the mouse and joystick data, and the keyboard
+ * interrupt is shared with the MIDI ACIA so MIDI data also get handled there).
+ * This driver only deals with handing key events off to the input layer.
+ *
+ * Largely based on the old:
+ *
+ * Atari Mouse Driver for Linux
+ * by Robert de Vries (robert@xxxxxx) 19Jul93
+ *
+ * 16 Nov 1994 Andreas Schwab
+ * Compatibility with busmouse
+ * Support for three button mouse (shamelessly stolen from MiNT)
+ * third button wired to one of the joystick directions on joystick 1
+ *
+ * 1996/02/11 Andreas Schwab
+ * Module support
+ * Allow multiple open's
+ *
+ * Converted to use new generic busmouse code. 5 Apr 1998
+ * Russell King <rmk@xxxxxxxxxxxxxxxxxxxx>
```


[patch 04/33] m68k: Atari keyboard and mouse support.

```
+ /* only relative events get here */
+ dx = buf[1];
+ dy = -buf[2];
+
+ input_report_rel(atamouse_dev, REL_X, dx);
+ input_report_rel(atamouse_dev, REL_Y, dy);
+
+ input_report_key(atamouse_dev, BTN_LEFT, buttons & 0x1);
+ input_report_key(atamouse_dev, BTN_MIDDLE, buttons & 0x2);
+ input_report_key(atamouse_dev, BTN_RIGHT, buttons & 0x4);
+
+ input_sync(atamouse_dev);
+
+ return;
+}
+
+static int atamouse_open(struct input_dev *dev)
+{
+ if (atamouse_used++)
+ return 0;
+
+ #ifdef FIXED_ATARI_JOYSTICK
+ atari_mouse_buttons = 0;
+ #endif
+ ikbd_mouse_y0_top();
+ ikbd_mouse_thresh(mouse_threshold[0], mouse_threshold[1]);
+ ikbd_mouse_rel_pos();
+ atari_input_mouse_interrupt_hook = atamouse_interrupt;
+ return 0;
+}
+
+static void atamouse_close(struct input_dev *dev)
+{
+ if (!--atamouse_used) {
+ ikbd_mouse_disable();
+ atari_mouse_interrupt_hook = NULL;
+ }
+}
+
+static int __init atamouse_init(void)
+{
+ if (!MACH_IS_ATARI || !ATARIHW_PRESENT(ST_MFP))
+ return -ENODEV;
+
+ if (!(atamouse_dev = input_allocate_device()))
+ return -ENOMEM;
+
+ if (!(atari_keyb_init()))
+ return -ENODEV;
+
+ atamouse_dev->name = "Atari mouse";
```

[patch 04/33] m68k: Atari keyboard and mouse support.

```
+ atamouse_dev->phys = "atamouse/input0";
+ atamouse_dev->id.bustype = BUS_ATARI;
+ atamouse_dev->id.vendor = 0x0001;
+ atamouse_dev->id.product = 0x0002;
+ atamouse_dev->id.version = 0x0100;
+
+ atamouse_dev->evbit[0] = BIT(EV_KEY) | BIT(EV_REL);
+ atamouse_dev->relbit[0] = BIT(REL_X) | BIT(REL_Y);
+ atamouse_dev->keybit[LONG(BTN_LEFT)] = BIT(BTN_LEFT) | BIT(BTN_MIDDLE) |
BIT(BTN_RIGHT);
+ atamouse_dev->open = atamouse_open;
+ atamouse_dev->close = atamouse_close;
+
+ input_register_device(atamouse_dev);
+
+ printk(KERN_INFO "input: %s at keyboard ACIA\n", atamouse_dev->name);
+ return 0;
+}
+
+static void __exit atamouse_exit(void)
+{
+ input_unregister_device(atamouse_dev);
+}
+
+module_init(atamouse_init);
+module_exit(atamouse_exit);
--- linux-m68k-2.6.21.orig/include/asm-m68k/atarikb.h
+++ linux-m68k-2.6.21/include/asm-m68k/atarikb.h
@@ -36,5 +36,11 @@ void ikbd_joystick_disable(void);
extern void (*atari_MIDI_interrupt_hook) (void);
/* Hook for mouse driver */
extern void (*atari_mouse_interrupt_hook) (char *);
+/* Hook for keyboard inputdev driver */
+extern void (*atari_input_keyboard_interrupt_hook) (unsigned char, char);
+/* Hook for mouse inputdev driver */
+extern void (*atari_input_mouse_interrupt_hook) (char *);
+
+int atari_keyb_init(void);

#endif /* _LINUX_ATARIKB_H */
--- linux-m68k-2.6.21.orig/include/linux/input.h
+++ linux-m68k-2.6.21/include/linux/input.h
@@ -676,6 +676,7 @@ struct input_absinfo {
#define BUS_I2C 0x18
#define BUS_HOST 0x19
#define BUS_GSC 0x1A
+#define BUS_ATARI 0x1B

/*
* Values describing the status of a force-feedback effect
```

[patch 04/33] m68k: Atari keyboard and mouse support.

—
Gr{oetje,eeting}s,

Geert

—
Geert Uytterhoeven — There's lots of Linux beyond ia32 — geert@xxxxxxxxxxxxxxxx

In personal conversations with technical people, I call myself a hacker. But
when I'm talking to journalists I just say "programmer" or something like that.
— Linus Torvalds

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