

>>>Signed-off-by: Marc St-Jean <Marc_St-Jean@xxxxxxxxxxxxxxxx>

>>>Index: linux_2_6/drivers/serial/8250.c

>>>=====

>>>RCS file: linux_2_6/drivers/serial/8250.c,v retrieving revision

>>>1.1.1.7 retrieving revision 1.9 diff -u -r1.1.1.7 -r1.9

>>>--- linux_2_6/drivers/serial/8250.c 19 Oct 2006 21:00:58

>>

>>-0000 1.1.1.7

>>

>>>+++ linux_2_6/drivers/serial/8250.c 19 Oct 2006 22:08:15

>>

>>-0000 1.9

>>

>>>@@ -44,6 +44,10 @@

>>> #include <asm/io.h>

>>> #include <asm/irq.h>

>>>

>>>+#ifdef CONFIG_PMC_MSP

>>>+#include <msp_regs.h>

>>>+#endif

>>>+

>>> #include "8250.h"

>>> /*

>>>@@ -130,6 +134,9 @@

>>> unsigned char mcr_mask; /* mask of user bits */

>>> unsigned char mcr_force; /* mask of forced bits */

>>> unsigned char lsr_break_flag;

>>>+#ifdef CONFIG_PMC_MSP

>>>+ int dwapb_lcr;

>> /* saved LCR for DW APB WAR */

>>>+#endif

>> There was already 'lcr' field there, couldn't it be used?

> Possibly but the current driver doesn't always save the LCR value before trying to write it. For example see serial8250_set_termios()

> serial_outp(up, UART_LCR, cval); /* reset DLAB */

> up->lcr = cval; /* Save LCR */

This certainly may be fixed.

> It's impossible to ensure that going forward the driver will always save the value before writing it.

> We need to have it saved before since the write will cause an

interrupt, hence the save in serial_out.

You may rework driver to always save it on write to 'lcr' field (in case of your UART). I don't think it's going to spoil something there...

> If that's not acceptable I can audit the driver and ensure all LCR writes are first saved.

What's **certainly** undesirable is #ifdef mess. :-)

```
>>>@@ -333,6 +340,10 @@
>>> static void
>>> serial_out(struct uart_8250_port *up, int offset, int value)
>>> {
>>>+#ifdef CONFIG_PMC_MSP
>>>+ /* Save the offset before it's remapped */
>>>+ int dwapb_offset = offset;
>>>+#endif
>>> offset = map_8250_out_reg(up, offset) << up->port.regshift;
>>>
>>> switch (up->port.iotype) {
>>>@@ -342,7 +353,19 @@
>>> break;
>>>
>>> case UPIO_MEM:
>>>+#ifdef CONFIG_PMC_MSP
>>>+ /* Save the LCR value so it can be re-written when a
>>>+ * Busy Detect interrupt occurs. */
>>>+ if (dwapb_offset == UART_LCR)
>>>+ up->dwapb_lcr = value;
>>>+#endif
>>> writeb(value, up->port.membase + offset);
>>>+#ifdef CONFIG_PMC_MSP
>>>+ /* Re-read the IER to ensure any interrupt disabling has
>>>+ * completed before proceeding with ISR. */
>>>+ if (dwapb_offset == UART_IER)
>>>+ value = serial_in(up, dwapb_offset); #endif
>>> break;
```

>> Hm, was there really a need for #ifdef mess here?

>> I'd vote for introducing new UPIO_* here, like was done

>>for TSi10x UARTs just for the same reason.

> I'm willing to do this, however IIRC there was a thread in late Sept. which came

> to the conclusion that UPIO were to be used for different access methods and not UART types.

AFAIR, almost every participant was left with his own opinion. I myself changed it twice during the discussion, to finally mostly agree with

Russell. :-D

> Do you see this as an exception?

Well, there's already an incident of using this to work around the register access errata (that UPIO_TSI I mentioned) — and I must note that this is certainly cleaner than #ifdef's, which should be avoided at all costs. :-)

> In the second #ifdef CONFIG_PMC_MSP above the workaround is required because of SoC

> interrupt design which is out-of-band with respect to r/w of UART registers, it's not

> specific to the DesignWare UART.

Erm... so why is this under #ifdef?

```
>>>-1141,6 +1175,12 @@
>>> iir = serial_in(up, UART_IIR);
>>> if (lsr & UART_LSR_TEMT && iir &
>>>
>>>UART_IIR_NO_INT)
>>>
>>> transmit_chars(up);
>>>+ } else if (up->bugs & UART_BUG_DWTHRE) {
>>>+ unsigned char lsr, iir;
>>>+ lsr = serial_in(up, UART_LSR);
>>>+ iir = serial_in(up, UART_IIR);
>>>+ if (lsr & UART_LSR_THRE)
>>>+ transmit_chars(up);
```

>> I don't see how this *really* differs from the UART_BUG_TXEN case.

>> Have you tried *that* workaround?

> I didn't write the code so I haven't tried it personally but I believe it was investigated.

> It looks like the results of the bugs are similar but the causes are different. In the UART_BUG_TXEN case,

> if I understand the comment correctly, NO interrupt is generated on enabling THRI.

> This is verified by looking for Transmitter Empty (0x40) and no interrupt.

I must note that "transmitter empty" condition in *not* the cause of the

THRI interrupt — it signifies that the TX shift register is empty, not the TX

holding register.

> In the UART_BUG_DWTHRE case an interrupt IS generated on enabling THRI.

If there was no prior sent characters, how it's generated?

- > However no new THRE interrupts will occur until a new character is written to the THR and it's sent.
- > This is verified by looking for Transmit-Hold-Register Empty (0x20).

Again, THRE condition doesn't mean that the character is actually sent. I only means that it's been stored to the shift register and so, the transmission started...

>> In any case, looks like this errata is auto-detectable just like UART_BUG_TXEN.

- > I don't know how we would be able to test this without sending junk test characters to the console port.

IIUC, you must *not* send characters to detect it. :-)

- > We currently enable the bug flag in our platform setup code when we know the bug is present from the SoC version.
- > Is that not acceptable?

Well, if that errata is indeed not readily detectable at runtime (or even effectively the same as already handled), this seems like the only option indeed...

```
>>>@@ -1366,6 +1406,31 @@
>>> handled = 1;
>>>
>>> end = NULL;
>>>+#ifdef CONFIG_PMC_MSP
>>>+ } else if ((iir & UART_IER_BUSY) == UART_IER_BUSY) {
```

>> Hm, masking IIR with IER mask, is this correct? Doubt it.

> I agree, that was badly named. I've changed it to UART_IIR_BUSY for the next spin.

```
>>>+ /*
>>>+ * The MSP (DesignWare APB UART) serial
>>>+ * subsystem has a
>>>+ * non-standard interrupt condition (0x7) which
>>>+ * means
>>>+ * that the LCR was written while the UART was
>>>+ * busy, so
>>>+ * the LCR was not actually written. It is
```

```
cleared by
>>>+ * reading the special non-standard extended
UART status
>>>+ * register.
>>>+ */
>>>+ unsigned int tmp;
>>>+ if( up->port.line == 0 )
>>>+ tmp = *UART0_STATUS_REG;
>>>+ else
>>>+ tmp = *UART1_STATUS_REG;
>>>+
>>>+ /* Check if saved on LCR write */
>>>+ if( up->dwapb_lcr != -1 )
>>>+ serial_outp(up, UART_LCR, up->dwapb_lcr);
>>>+ else
>>>+ printk(KERN_ERR "serial8250: UART BUSY,
no LCR write!\n" );
>>>+
>>>+ handled = 1;
>>>+ end = NULL;
>>>+#endif
```

>> Not sure if this also shouldn't be handled in other
>>places which check for interrupt status, like serial8250_timeout()...

> I can move the code to a function but I still see two issues:
> A) We could move the code the a new function. We could also check for
a new UART_BUG_* flag before calling the function,

> but in reality this is a feature of the UART not a bug.

May also check for certain UPIO_* once it's introduced...

> B) How to eliminate the platform #ifdef. How to pass in the address
of the UARTx_STATUS_REG in a platform independent way?

I'd consider defining the UART status reg. in serial_reg.h and also
doing
reads via serial_in(),
again under a new UPIO_* case...

> We could add it to one of the data structures like uart_port, but it
would need an #ifdef in the header file.

Add what, register?

```
>>>Index: linux_2_6/include/linux/serial_reg.h
>>>=====
>>>RCS file: linux_2_6/include/linux/serial_reg.h,v
>>>retrieving revision 1.1.1.2
>>>retrieving revision 1.3
```

Re: [PATCH] serial driver PMC MSP71xx, kernel linux-mips.git mast er

```
>>>diff -u -r1.1.1.2 -r1.3
>>>--- linux_2_6/include/linux/serial_reg.h 19 Oct 2006 18:29:50
-0000 1.1.1.2
>>>+++ linux_2_6/include/linux/serial_reg.h 19 Oct 2006 19:45:04
-0000 1.3
>>>@@ -218,6 +218,10 @@
>>> #define UART_FCR_PXAR16 0x80 /* receive FIFO treshold = 16 */
>>> #define UART_FCR_PXAR32 0xc0 /* receive FIFO treshold = 32 */

>>>+/*
>>>+ * DesignWare APB UART
>>>+ */
>>>+#define UART_IER_BUSY 0x07 /* Busy Detect */
```

>> Are you sure it's not *IIR* value? Doesn't look like
>>interrupt mask for IER. And IIR value of 7 already means
>>something else, namely, no interrupt and receiver status. Hm...

> As mentioned earlier I have now renamed this UART_IIR_BUSY.

> From serial_reg.h, the receiver status is 0x06 and no interrupt is
0x01. I thought these couldn't both be set at the same time?

> In any case this is how DesignWare implemented the status so we can't
change it now.

Well, bits 1-2 have no meaning when bit 0 is set. Probably, chip
designers
decided to abuse this. :-)

> Thanks for the feedback,
> Marc

MBR, Sergei

-

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/>

Re: [PATCH] serial driver PMC MSP71xx, kernel linux-mips.git mast er