```
**********************************************************
   34729 Sun Mar  2 12:33:18 2014
new/usr/src/uts/i86pc/io/pcplusmp/apic.c
4665 pcplusmp open-codes register operations
**********************************************************
______unchanged_portion_omitted_
```

```
529 #endif /* DEBUG */

531 /*
532  * platform_intr_enter
533  *
534  *      Called at the beginning of the interrupt service routine to
535  *      mask all level equal to and below the interrupt priority
536  *      of the interrupting vector.  An EOI should be given to
537  *      the interrupt controller to enable other HW interrupts.
538  *
539  *      Return -1 for spurious interrupts
540  *
541  */
542 /*ARGSUSED*/
543 static int
544 apic_intr_enter(int ipl, int *vectorp)
545 {
546         uchar_t vector;
547         int nipl;
548         int irq;
549         ulong_t iflag;
550         apic_cpus_info_t *cpu_infop;

552         /*
553          * The real vector delivered is (*vectorp + 0x20), but our caller
554          * subtracts 0x20 from the vector before passing it to us.
555          * (That's why APIC_BASE_VECT is 0x20.)
556          */
557         vector = (uchar_t)*vectorp;

559         /* if interrupted by the clock, increment apic_nsec_since_boot */
560         if (vector == apic_clkvect) {
561                 if (!apic_oneshot) {
562                         /* NOTE: this is not MT aware */
563                         apic_hrtime_stamp++;
564                         apic_nsec_since_boot += apic_nsec_per_intr;
565                         apic_hrtime_stamp++;
566                         last_count_read = apic_hertz_count;
567                         apic_redistribute_compute();
568                 }

570                 /* We will avoid all the book keeping overhead for clock */
571                 nipl = apic_ipls[vector];

573                 *vectorp = apic_vector_to_irq[vector + APIC_BASE_VECT];

575                 apic_reg_ops->apic_write_task_reg(apic_ipltopri[nipl]);
576                 apic_reg_ops->apic_send_eoi(0);
574                 if (apic_mode == LOCAL_APIC) {
575 #if defined(__amd64)
576                         setcr8((ulong_t)(apic_ipltopri[nipl] >>
577                             APIC_IPL_SHIFT));
578 #else
579                         if (apic_have_32bit_cr8)
580                                 setcr8((ulong_t)(apic_ipltopri[nipl] >>
581                                     APIC_IPL_SHIFT));
582                         else
583                                 LOCAL_APIC_WRITE_REG(APIC_TASK_REG,
584                                     (uint32_t)apic_ipltopri[nipl]);
585 #endif
```



```
586                         LOCAL_APIC_WRITE_REG(APIC_EOI_REG, 0);
587                 } else {
588                         X2APIC_WRITE(APIC_TASK_REG, apic_ipltopri[nipl]);
589                         X2APIC_WRITE(APIC_EOI_REG, 0);
590                 }

578                 return (nipl);
579         }

581         cpu_infop = &apic_cpus[psm_get_cpu_id()];

583         if (vector == (APIC_SPUR_INTR - APIC_BASE_VECT)) {
584                 cpu_infop->aci_spur_cnt++;
585                 return (APIC_INT_SPURIOUS);
586         }

588         /* Check if the vector we got is really what we need */
589         if (apic_revector_pending) {
590                 /*
591                  * Disable interrupts for the duration of
592                  * the vector translation to prevent a self-race for
593                  * the apic_revector_lock.  This cannot be done
594                  * in apic_xlate_vector because it is recursive and
595                  * we want the vector translation to be atomic with
596                  * respect to other (higher-priority) interrupts.
597                  */
598                 iflag = intr_clear();
599                 vector = apic_xlate_vector(vector + APIC_BASE_VECT) -
600                     APIC_BASE_VECT;
601                 intr_restore(iflag);
602         }

604         nipl = apic_ipls[vector];
605         *vectorp = irq = apic_vector_to_irq[vector + APIC_BASE_VECT];

607         apic_reg_ops->apic_write_task_reg(apic_ipltopri[nipl]);
621         if (apic_mode == LOCAL_APIC) {
622 #if defined(__amd64)
623                 setcr8((ulong_t)(apic_ipltopri[nipl] >> APIC_IPL_SHIFT));
624 #else
625                 if (apic_have_32bit_cr8)
626                         setcr8((ulong_t)(apic_ipltopri[nipl] >>
627                             APIC_IPL_SHIFT));
628                 else
629                         LOCAL_APIC_WRITE_REG(APIC_TASK_REG,
630                             (uint32_t)apic_ipltopri[nipl]);
631 #endif
632         } else {
633                 X2APIC_WRITE(APIC_TASK_REG, apic_ipltopri[nipl]);
634         }

609         cpu_infop->aci_current[nipl] = (uchar_t)irq;
610         cpu_infop->aci_curipl = (uchar_t)nipl;
611         cpu_infop->aci_ISR_in_progress |= 1 << nipl;

613         /*
614          * apic_level_intr could have been assimilated into the irq struct.
615          * but, having it as a character array is more efficient in terms of
616          * cache usage. So, we leave it as is.
617          */
618         if (!apic_level_intr[irq]) {
619                 apic_reg_ops->apic_send_eoi(0);
646                 if (apic_mode == LOCAL_APIC) {
647                         LOCAL_APIC_WRITE_REG(APIC_EOI_REG, 0);
648                 } else {
649                         X2APIC_WRITE(APIC_EOI_REG, 0);
```

```
650                 }
620         }

622 #ifdef  DEBUG
623         APIC_DEBUG_BUF_PUT(vector);
624         APIC_DEBUG_BUF_PUT(irq);
625         APIC_DEBUG_BUF_PUT(nipl);
626         APIC_DEBUG_BUF_PUT(psm_get_cpu_id());
627         if ((apic_stretch_interrupts) && (apic_stretch_ISR & (1 << nipl)))
628                 drv_usecwait(apic_stretch_interrupts);

630         if (apic_break_on_cpu == psm_get_cpu_id())
631                 apic_break();
632 #endif /* DEBUG */
633         return (nipl);
634 }
```

_____unchanged_portion_omitted_

```
650 /*
651  * Any changes made to this function must also change X2APIC
652  * version of intr_exit.
653  */
654 void
655 apic_intr_exit(int prev_ipl, int irq)
656 {
657         apic_cpus_info_t *cpu_infop;

659         apic_reg_ops->apic_write_task_reg(apic_ipltopri[prev_ipl]);
690 #if defined(__amd64)
691         setcr8((ulong_t)(apic_ipltopri[prev_ipl] >> APIC_IPL_SHIFT));
692 #else
693         if (apic_have_32bit_cr8)
694                 setcr8((ulong_t)(apic_ipltopri[prev_ipl] >> APIC_IPL_SHIFT));
695         else
696                 apicadr[APIC_TASK_REG] = apic_ipltopri[prev_ipl];
697 #endif

661         APIC_INTR_EXIT();
662 }
```

_____unchanged_portion_omitted_

```
686 /*
687  * Mask all interrupts below or equal to the given IPL.
688  * Any changes made to this function must also change X2APIC
689  * version of setspl.
690  */
691 static void
692 apic_setspl(int ipl)
693 {
694         apic_reg_ops->apic_write_task_reg(apic_ipltopri[ipl]);
732 #if defined(__amd64)
733         setcr8((ulong_t)(apic_ipltopri[ipl] >> APIC_IPL_SHIFT));
734 #else
735         if (apic_have_32bit_cr8)
736                 setcr8((ulong_t)(apic_ipltopri[ipl] >> APIC_IPL_SHIFT));
737         else
738                 apicadr[APIC_TASK_REG] = apic_ipltopri[ipl];
739 #endif

696         /* interrupts at ipl above this cannot be in progress */
697         apic_cpus[psm_get_cpu_id()].aci_ISR_in_progress &= (2 << ipl) - 1;
698         /*
699          * this is a patch fix for the ALR QSMP P5 machine, so that interrupts
700          * have enough time to come in before the priority is raised again
701          * during the idle() loop.
702          */
```



```
703         if (apic_setspl_delay)
704                 (void) apic_reg_ops->apic_get_pri();
705 }
```

_____unchanged_portion_omitted_