```
**********************************************************
    1978 Thu Sep  7 15:25:31 2017
new/usr/src/uts/i86pc/apix/Makefile
8626 make pcplusmp and apix warning-free
Reviewed by: Robert Mustacchi <rm@joyent.com>
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
#
# uts/i86pc/apix/Makefile
#
# Copyright (c) 2010, Oracle and/or its affiliates. All rights reserved.
# Copyright 2016, Joyent, Inc.
#
#       This makefile drives the production of the pcplusmp "mach"
#       kernel module.
#
#       pcplusmp implementation architecture dependent
#

#
#       Path to the base of the uts directory tree (usually /usr/src/uts).
#
UTSBASE = ../..

#
#       Define the module and object file sets.
#
MODULE          = apix
OBJECTS         = $(APIX_OBJS:%=$(OBJS_DIR)/%)
LINTS           = $(APIX_OBJS:%.o=$(LINTS_DIR)/%.ln)
ROOTMODULE      = $(ROOT_PSM_MACH_DIR)/$(MODULE)

#
#       Include common rules.
#
include $(UTSBASE)/i86pc/Makefile.i86pc

#
#       Define targets
#
ALL_TARGET      = $(BINARY)
LINT_TARGET     = $(MODULE).lint
INSTALL_TARGET  = $(BINARY) $(ROOTMODULE)

DEBUG_FLGS      =
$(NOT_RELEASE_BUILD)DEBUG_DEFS  += $(DEBUG_FLGS)
```



```
#
# Depends on ACPI CA interpreter
#
LDFLAGS         += -dy -N misc/acpica

CERRWARN        += -_gcc=-Wno-uninitialized
CERRWARN        += -_gcc=-Wno-unused-function

#
#       Default build targets.
#
.KEEP_STATE:

def:            $(DEF_DEPS)

all:            $(ALL_DEPS)

clean:          $(CLEAN_DEPS)

clobber:        $(CLOBBER_DEPS)

lint:           $(LINT_DEPS)

modlintlib:     $(MODLINTLIB_DEPS)

clean.lint:     $(CLEAN_LINT_DEPS)

install:        $(INSTALL_DEPS)

#
#       Include common targets.
#
include $(UTSBASE)/i86pc/Makefile.targ
```

```
**********************************************************
   65927 Thu Sep  7 15:25:31 2017
new/usr/src/uts/i86pc/io/apix/apix.c
8626 make pcplusmp and apix warning-free
Reviewed by: Robert Mustacchi <rm@joyent.com>
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
**********************************************************
______unchanged_portion_omitted_

1600 apix_vector_t *
1601 apix_set_cpu(apix_vector_t *vecp, int new_cpu, int *result)
1602 {
1603         apix_vector_t *newp = NULL;
1604         dev_info_t *dip;
1605         int inum, cap_ptr;
1606         ddi_acc_handle_t handle;
1607         ddi_intr_msix_t *msix_p = NULL;
1608         ushort_t msix_ctrl;
1609         uintptr_t off = 0;
1610         uint32_t mask = 0;
1609         uintptr_t off;
1610         uint32_t mask;

1612         ASSERT(LOCK_HELD(&apix_lock));
1613         *result = ENXIO;

1615         /* Fail if this is an MSI intr and is part of a group. */
1616         if (vecp->v_type == APIX_TYPE_MSI) {
1617                 if (i_ddi_intr_get_current_nintrs(APIX_GET_DIP(vecp)) > 1)
1618                         return (NULL);
1619                 else
1620                         return (apix_grp_set_cpu(vecp, new_cpu, result));
1621         }

1623         /*
1624          * Mask MSI-X. It's unmasked when MSI-X gets enabled.
1625          */
1626         if (vecp->v_type == APIX_TYPE_MSIX && IS_VECT_ENABLED(vecp)) {
1627                 if ((dip = APIX_GET_DIP(vecp)) == NULL)
1628                         return (NULL);
1629                 inum = vecp->v_devp->dv_inum;

1631                 handle = i_ddi_get_pci_config_handle(dip);
1632                 cap_ptr = i_ddi_get_msi_msix_cap_ptr(dip);
1633                 msix_ctrl = pci_config_get16(handle, cap_ptr + PCI_MSIX_CTRL);
1634                 if ((msix_ctrl & PCI_MSIX_FUNCTION_MASK) == 0) {
1635                         /*
1636                          * Function is not masked, then mask "inum"th
1637                          * entry in the MSI-X table
1638                          */
1639                         msix_p = i_ddi_get_msix(dip);
1640                         off = (uintptr_t)msix_p->msix_tbl_addr + (inum *
1641                             PCI_MSIX_VECTOR_SIZE) + PCI_MSIX_VECTOR_CTRL_OFFSET;
1642                         mask = ddi_get32(msix_p->msix_tbl_hdl, (uint32_t *)off);
1643                         ddi_put32(msix_p->msix_tbl_hdl, (uint32_t *)off,
1644                             mask | 1);
1645                 }
1646         }

1648         *result = 0;
1649         if ((newp = apix_rebind(vecp, new_cpu, 1)) == NULL)
1650                 *result = EIO;

1652         /* Restore mask bit */
1653         if (msix_p != NULL)
1654                 ddi_put32(msix_p->msix_tbl_hdl, (uint32_t *)off, mask);
```



```
1656         return (newp);
1657 }

1659 /*
1660  * Set cpu for MSIs
1661  */
1662 apix_vector_t *
1663 apix_grp_set_cpu(apix_vector_t *vecp, int new_cpu, int *result)
1664 {
1665         apix_vector_t *newp, *vp;
1666         uint32_t orig_cpu = vecp->v_cpuid;
1667         int orig_vect = vecp->v_vector;
1668         int i, num_vectors, cap_ptr, msi_mask_off = 0;
1669         uint32_t msi_pvm = 0;
1668         int i, num_vectors, cap_ptr, msi_mask_off;
1669         uint32_t msi_pvm;
1670         ushort_t msi_ctrl;
1671         ddi_acc_handle_t handle;
1672         dev_info_t *dip;

1674         APIC_VERBOSE(INTR, (CE_CONT, "apix_grp_set_cpu: oldcpu: %x, vector: %x,"
1675             " newcpu:%x\n", vecp->v_cpuid, vecp->v_vector, new_cpu));

1677         ASSERT(LOCK_HELD(&apix_lock));

1679         *result = ENXIO;

1681         if (vecp->v_type != APIX_TYPE_MSI) {
1682                 DDI_INTR_IMPLDBG((CE_WARN, "set_grp: intr not MSI\n"));
1683                 return (NULL);
1684         }

1686         if ((dip = APIX_GET_DIP(vecp)) == NULL)
1687                 return (NULL);

1689         num_vectors = i_ddi_intr_get_current_nintrs(dip);
1690         if ((num_vectors < 1) || ((num_vectors - 1) & orig_vect)) {
1691                 APIC_VERBOSE(INTR, (CE_WARN,
1692                     "set_grp: base vec not part of a grp or not aligned: "
1693                     "vec:0x%x, num_vec:0x%x\n", orig_vect, num_vectors));
1694                 return (NULL);
1695         }

1697         if (vecp->v_inum != apix_get_min_dev_inum(dip, vecp->v_type))
1698                 return (NULL);

1700         *result = EIO;
1701         for (i = 1; i < num_vectors; i++) {
1702                 if ((vp = xv_vector(orig_cpu, orig_vect + i)) == NULL)
1703                         return (NULL);
1704 #ifdef DEBUG
1705                 /*
1706                  * Sanity check: CPU and dip is the same for all entries.
1707                  * May be called when first msi to be enabled, at this time
1708                  * add_avintr() is not called for other msi
1709                  */
1710                 if ((vp->v_share != 0) &&
1711                     ((APIX_GET_DIP(vp) != dip) ||
1712                     (vp->v_cpuid != vecp->v_cpuid))) {
1713                         APIC_VERBOSE(INTR, (CE_WARN,
1714                             "set_grp: cpu or dip for vec 0x%x difft than for "
1715                             "vec 0x%x\n", orig_vect, orig_vect + i));
1716                         APIC_VERBOSE(INTR, (CE_WARN,
1717                             "  cpu: %d vs %d, dip: 0x%p vs 0x%p\n", orig_cpu,
1718                             vp->v_cpuid, (void *)dip,
```

```
1719                            (void *)APIX_GET_DIP(vp)));
1720                        return (NULL);
1721                }
1722 #endif /* DEBUG */
1723         }

1725         cap_ptr = i_ddi_get_msi_msix_cap_ptr(dip);
1726         handle = i_ddi_get_pci_config_handle(dip);
1727         msi_ctrl = pci_config_get16(handle, cap_ptr + PCI_MSI_CTRL);

1729         /* MSI Per vector masking is supported. */
1730         if (msi_ctrl & PCI_MSI_PVM_MASK) {
1731                 if (msi_ctrl &  PCI_MSI_64BIT_MASK)
1732                         msi_mask_off = cap_ptr + PCI_MSI_64BIT_MASKBITS;
1733                 else
1734                         msi_mask_off = cap_ptr + PCI_MSI_32BIT_MASK;
1735                 msi_pvm = pci_config_get32(handle, msi_mask_off);
1736                 pci_config_put32(handle, msi_mask_off, (uint32_t)-1);
1737                 APIC_VERBOSE(INTR, (CE_CONT,
1738                     "set_grp: pvm supported.  Mask set to 0x%x\n",
1739                     pci_config_get32(handle, msi_mask_off)));
1740         }

1742         if ((newp = apix_rebind(vecp, new_cpu, num_vectors)) != NULL)
1743                 *result = 0;

1745         /* Reenable vectors if per vector masking is supported. */
1746         if (msi_ctrl & PCI_MSI_PVM_MASK) {
1747                 pci_config_put32(handle, msi_mask_off, msi_pvm);
1748                 APIC_VERBOSE(INTR, (CE_CONT,
1749                     "set_grp: pvm supported.  Mask restored to 0x%x\n",
1750                     pci_config_get32(handle, msi_mask_off)));
1751         }

1753         return (newp);
1754 }
```

*_____unchanged_portion_omitted_*

```
**********************************************************
   68957 Thu Sep  7 15:25:32 2017
new/usr/src/uts/i86pc/io/mp_platform_common.c
8626 make pcplusmp and apix warning-free
Reviewed by: Robert Mustacchi <rm@joyent.com>
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright (c) 2007, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright 2016 Nexenta Systems, Inc.
 * Copyright (c) 2017 by Delphix. All rights reserved.
 * Copyright 2017 Joyent, Inc.
 */
/*
 * Copyright (c) 2010, Intel Corporation.
 * All rights reserved.
 */

/*
 * PSMI 1.1 extensions are supported only in 2.6 and later versions.
 * PSMI 1.2 extensions are supported only in 2.7 and later versions.
 * PSMI 1.3 and 1.4 extensions are supported in Solaris 10.
 * PSMI 1.5 extensions are supported in Solaris Nevada.
 * PSMI 1.6 extensions are supported in Solaris Nevada.
 * PSMI 1.7 extensions are supported in Solaris Nevada.
 */
#define PSMI_1_7

#include <sys/processor.h>
#include <sys/time.h>
#include <sys/psm.h>
#include <sys/smp_impldefs.h>
#include <sys/cram.h>
#include <sys/acpi/acpi.h>
#include <sys/acpica.h>
#include <sys/psm_common.h>
#include <sys/apic.h>
#include <sys/apic_timer.h>
#include <sys/pit.h>
#include <sys/ddi.h>
#include <sys/sunddi.h>
#include <sys/ddi_impldefs.h>
#include <sys/pci.h>
#include <sys/promif.h>
#include <sys/x86_archext.h>
#include <sys/cpc_impl.h>
```



```
#include <sys/uadmin.h>
#include <sys/panic.h>
#include <sys/debug.h>
#include <sys/archsystm.h>
#include <sys/trap.h>
#include <sys/machsystm.h>
#include <sys/cpuvar.h>
#include <sys/rm_platter.h>
#include <sys/privregs.h>
#include <sys/cyclic.h>
#include <sys/note.h>
#include <sys/pci_intr_lib.h>
#include <sys/sunndi.h>
#if !defined(__xpv)
#include <sys/hpet.h>
#include <sys/clock.h>
#endif

/*
 *      Local Function Prototypes
 */
static int apic_handle_defconf();
static int apic_parse_mpct(caddr_t mpct, int bypass);
static struct apic_mpfps_hdr *apic_find_fps_sig(caddr_t fptr, int size);
static int apic_checksum(caddr_t bptr, int len);
static int apic_find_bus_type(char *bus);
static int apic_find_bus(int busid);
static struct apic_io_intr *apic_find_io_intr(int irqno);
static int apic_find_free_irq(int start, int end);
struct apic_io_intr *apic_find_io_intr_w_busid(int irqno, int busid);
static void apic_set_pwroff_method_from_mpcnfhdr(struct apic_mp_cnf_hdr *hdrp);
static void apic_free_apic_cpus(void);
static boolean_t apic_is_ioapic_AMD_813x(uint32_t physaddr);
static int apic_acpi_enter_apicmode(void);

int apic_handle_pci_pci_bridge(dev_info_t *idip, int child_devno,
    int child_ipin, struct apic_io_intr **intrp);
int apic_find_bus_id(int bustype);
int apic_find_intin(uchar_t ioapic, uchar_t intin);
void apic_record_rdt_entry(apic_irq_t *irqptr, int irq);

int apic_debug_mps_id = 0;      /* 1 - print MPS ID strings */

/* ACPI SCI interrupt configuration; -1 if SCI not used */
int apic_sci_vect = -1;
iflag_t apic_sci_flags;

#if !defined(__xpv)
/* ACPI HPET interrupt configuration; -1 if HPET not used */
int apic_hpet_vect = -1;
iflag_t apic_hpet_flags;
#endif

/*
 * psm name pointer
 */
char *psm_name;

/* ACPI support routines */
static int acpi_probe(char *);
static int apic_acpi_irq_configure(acpi_psm_lnk_t *acpipsmlnkp, dev_info_t *dip,
    int *pci_irqp, iflag_t *intr_flagp);

int apic_acpi_translate_pci_irq(dev_info_t *dip, int busid, int devid,
    int ipin, int *pci_irqp, iflag_t *intr_flagp);
uchar_t acpi_find_ioapic(int irq);
```

```
126 static int acpi_intr_compatible(iflag_t iflag1, iflag_t iflag2);

128 /* Max wait time (in repetitions) for flags to clear in an RDT entry. */
129 int apic_max_reps_clear_pending = 1000;

131 int     apic_intr_policy = INTR_ROUND_ROBIN;

133 int     apic_next_bind_cpu = 1; /* For round robin assignment */
134                                 /* start with cpu 1 */

136 /*
137  * If enabled, the distribution works as follows:
138  * On every interrupt entry, the current ipl for the CPU is set in cpu_info
139  * and the irq corresponding to the ipl is also set in the aci_current array.
140  * interrupt exit and setspl (due to soft interrupts) will cause the current
141  * ipl to be be changed. This is cache friendly as these frequently used
142  * paths write into a per cpu structure.
143  *
144  * Sampling is done by checking the structures for all CPUs and incrementing
145  * the busy field of the irq (if any) executing on each CPU and the busy field
146  * of the corresponding CPU.
147  * In periodic mode this is done on every clock interrupt.
148  * In one-shot mode, this is done thru a cyclic with an interval of
149  * apic_redistribute_sample_interval (default 10 milli sec).
150  *
151  * Every apic_sample_factor_redistribution times we sample, we do computations
152  * to decide which interrupt needs to be migrated (see comments
153  * before apic_intr_redistribute().
154  */

156 /*
157  * Following 3 variables start as % and can be patched or set using an
158  * API to be defined in future. They will be scaled to
159  * sample_factor_redistribution which is in turn set to hertz+1 (in periodic
160  * mode), or 101 in one-shot mode to stagger it away from one sec processing
161  */

163 int     apic_int_busy_mark = 60;
164 int     apic_int_free_mark = 20;
165 int     apic_diff_for_redistribution = 10;

167 /* sampling interval for interrupt redistribution for dynamic migration */
168 int     apic_redistribute_sample_interval = NANOSEC / 100; /* 10 millisec */

170 /*
171  * number of times we sample before deciding to redistribute interrupts
172  * for dynamic migration
173  */
174 int     apic_sample_factor_redistribution = 101;

176 int     apic_redist_cpu_skip = 0;
177 int     apic_num_imbalance = 0;
178 int     apic_num_rebind = 0;

180 /*
181  * Maximum number of APIC CPUs in the system, -1 indicates that dynamic
182  * allocation of CPU ids is disabled.
183  */
184 int     apic_max_nproc = -1;
185 int     apic_nproc = 0;
186 size_t  apic_cpus_size = 0;
187 int     apic_defconf = 0;
188 int     apic_irq_translate = 0;
189 int     apic_spec_rev = 0;
190 int     apic_imcrp = 0;
```



```
192 int     apic_use_acpi = 1;      /* 1 = use ACPI, 0 = don't use ACPI */
193 int     apic_use_acpi_madt_only = 0;    /* 1=ONLY use MADT from ACPI */

195 /*
196  * For interrupt link devices, if apic_unconditional_srs is set, an irq resource
197  * will be assigned (via _SRS). If it is not set, use the current
198  * irq setting (via _CRS), but only if that irq is in the set of possible
199  * irqs (returned by _PRS) for the device.
200  */
201 int     apic_unconditional_srs = 1;

203 /*
204  * For interrupt link devices, if apic_prefer_crs is set when we are
205  * assigning an IRQ resource to a device, prefer the current IRQ setting
206  * over other possible irq settings under same conditions.
207  */

209 int     apic_prefer_crs = 1;

211 uchar_t apic_io_id[MAX_IO_APIC];
212 volatile uint32_t *apicioadr[MAX_IO_APIC];
213 uchar_t apic_io_ver[MAX_IO_APIC];
214 uchar_t apic_io_vectbase[MAX_IO_APIC];
215 uchar_t apic_io_vectend[MAX_IO_APIC];
216 uchar_t apic_reserved_irqlist[MAX_ISA_IRQ + 1];
217 uint32_t apic_physaddr[MAX_IO_APIC];

219 boolean_t ioapic_mask_workaround[MAX_IO_APIC];

221 /*
222  * First available slot to be used as IRQ index into the apic_irq_table
223  * for those interrupts (like MSI/X) that don't have a physical IRQ.
224  */
225 int apic_first_avail_irq  = APIC_FIRST_FREE_IRQ;

227 /*
228  * apic_ioapic_lock protects the ioapics (reg select), the status, temp_bound
229  * and bound elements of cpus_info and the temp_cpu element of irq_struct
230  */
231 lock_t  apic_ioapic_lock;

233 int     apic_io_max = 0;        /* no. of i/o apics enabled */

235 struct apic_io_intr *apic_io_intrp = NULL;
236 static  struct apic_bus *apic_busp;

238 uchar_t apic_resv_vector[MAXIPL+1];

240 char    apic_level_intr[APIC_MAX_VECTOR+1];

242 uint32_t        eisa_level_intr_mask = 0;
243         /* At least MSB will be set if EISA bus */

245 int     apic_pci_bus_total = 0;
246 uchar_t apic_single_pci_busid = 0;

248 /*
249  * airq_mutex protects additions to the apic_irq_table - the first
250  * pointer and any airq_nexts off of that one. It also protects
251  * apic_max_device_irq & apic_min_device_irq. It also guarantees
252  * that share_id is unique as new ids are generated only when new
253  * irq_t structs are linked in. Once linked in the structs are never
254  * deleted. temp_cpu & mps_intr_index field indicate if it is programmed
255  * or allocated. Note that there is a slight gap between allocating in
256  * apic_introp_xlate and programming in addspl.
257  */
```

```
258 kmutex_t        airq_mutex;
259 apic_irq_t      *apic_irq_table[APIC_MAX_VECTOR+1];
260 int             apic_max_device_irq = 0;
261 int             apic_min_device_irq = APIC_MAX_VECTOR;

263 typedef struct prs_irq_list_ent {
264         int                     list_prio;
265         int32_t                 irq;
266         iflag_t                 intrflags;
267         acpi_prs_private_t      prsprv;
268         struct prs_irq_list_ent *next;
269 } prs_irq_list_t;
_____unchanged_portion_omitted_

311 int     apic_poweroff_method = APIC_POWEROFF_NONE;

313 /*
314  * Auto-configuration routines
315  */

317 /*
318  * Look at MPSpec 1.4 (Intel Order # 242016-005) for details of what we do here
319  * May work with 1.1 - but not guaranteed.
320  * According to the MP Spec, the MP floating pointer structure
321  * will be searched in the order described below:
322  * 1. In the first kilobyte of Extended BIOS Data Area (EBDA)
323  * 2. Within the last kilobyte of system base memory
324  * 3. In the BIOS ROM address space between 0F0000h and 0FFFFh
325  * Once we find the right signature with proper checksum, we call
326  * either handle_defconf or parse_mpct to get all info necessary for
327  * subsequent operations.
328  */
329 int
330 apic_probe_common(char *modname)
331 {
332         uint32_t mpct_addr, ebda_start = 0, base_mem_end;
333         caddr_t biosdatap;
334         caddr_t mpct = NULL;
333         caddr_t mpct = 0;
335         caddr_t fptr;
336         int     i, mpct_size = 0, mapsize, retval = PSM_FAILURE;
335         int     i, mpct_size, mapsize, retval = PSM_FAILURE;
337         ushort_t        ebda_seg, base_mem_size;
338         struct  apic_mpfps_hdr  *fpsp;
339         struct  apic_mp_cnf_hdr *hdrp;
340         int bypass_cpu_and_ioapics_in_mptables;
341         int acpi_user_options;

343         if (apic_forceload < 0)
344                 return (retval);

346         /*
347          * Remember who we are
348          */
349         psm_name = modname;

351         /* Allow override for MADT-only mode */
352         acpi_user_options = ddi_prop_get_int(DDI_DEV_T_ANY, ddi_root_node(), 0,
353             "acpi-user-options", 0);
354         apic_use_acpi_madt_only = ((acpi_user_options & ACPI_OUSER_MADT) != 0);

356         /* Allow apic_use_acpi to override MADT-only mode */
357         if (!apic_use_acpi)
358                 apic_use_acpi_madt_only = 0;

360         retval = acpi_probe(modname);
```



```
362         /* in UEFI system, there is no BIOS data */
363         if (ddi_prop_exists(DDI_DEV_T_ANY, ddi_root_node(), 0, "efi-systab"))
364                 goto apic_ret;

366         /*
367          * mapin the bios data area 40:0
368          * 40:13h - two-byte location reports the base memory size
369          * 40:0Eh - two-byte location for the exact starting address of
370          *          the EBDA segment for EISA
371          */
372         biosdatap = psm_map_phys(0x400, 0x20, PROT_READ);
373         if (!biosdatap)
374                 goto apic_ret;
375         fpsp = (struct apic_mpfps_hdr *)NULL;
376         mapsize = MPFPS_RAM_WIN_LEN;
377         /*LINTED: pointer cast may result in improper alignment */
378         ebda_seg = *((ushort_t *)(biosdatap+0xe));
379         /* check the 1k of EBDA */
380         if (ebda_seg) {
381                 ebda_start = ((uint32_t)ebda_seg) << 4;
382                 fptr = psm_map_phys(ebda_start, MPFPS_RAM_WIN_LEN, PROT_READ);
383                 if (fptr) {
384                         if (!(fpsp =
385                             apic_find_fps_sig(fptr, MPFPS_RAM_WIN_LEN)))
386                                 psm_unmap_phys(fptr, MPFPS_RAM_WIN_LEN);
387                 }
388         }
389         /* If not in EBDA, check the last k of system base memory */
390         if (!fpsp) {
391                 /*LINTED: pointer cast may result in improper alignment */
392                 base_mem_size = *((ushort_t *)(biosdatap + 0x13));

394                 if (base_mem_size > 512)
395                         base_mem_end = 639 * 1024;
396                 else
397                         base_mem_end = 511 * 1024;
398                 /* if ebda == last k of base mem, skip to check BIOS ROM */
399                 if (base_mem_end != ebda_start) {

401                         fptr = psm_map_phys(base_mem_end, MPFPS_RAM_WIN_LEN,
402                             PROT_READ);

404                         if (fptr) {
405                                 if (!(fpsp = apic_find_fps_sig(fptr,
406                                     MPFPS_RAM_WIN_LEN)))
407                                         psm_unmap_phys(fptr, MPFPS_RAM_WIN_LEN);
408                         }
409                 }
410         }
411         psm_unmap_phys(biosdatap, 0x20);

413         /* If still cannot find it, check the BIOS ROM space */
414         if (!fpsp) {
415                 mapsize = MPFPS_ROM_WIN_LEN;
416                 fptr = psm_map_phys(MPFPS_ROM_WIN_START,
417                     MPFPS_ROM_WIN_LEN, PROT_READ);
418                 if (fptr) {
419                         if (!(fpsp =
420                             apic_find_fps_sig(fptr, MPFPS_ROM_WIN_LEN))) {
421                                 psm_unmap_phys(fptr, MPFPS_ROM_WIN_LEN);
422                                 goto apic_ret;
423                         }
424                 }
425         }
```

```
427         if (apic_checksum((caddr_t)fpsp, fpsp->mpfps_length * 16) != 0) {
428                 psm_unmap_phys(fptr, MPFPS_ROM_WIN_LEN);
429                 goto apic_ret;
430         }

432         apic_spec_rev = fpsp->mpfps_spec_rev;
433         if ((apic_spec_rev != 04) && (apic_spec_rev != 01)) {
434                 psm_unmap_phys(fptr, MPFPS_ROM_WIN_LEN);
435                 goto apic_ret;
436         }

438         /* check IMCR is present or not */
439         apic_imcrp = fpsp->mpfps_featinfo2 & MPFPS_FEATINFO2_IMCRP;

441         /* check default configuration (dual CPUs) */
442         if ((apic_defconf = fpsp->mpfps_featinfo1) != 0) {
443                 psm_unmap_phys(fptr, mapsize);
444                 if ((retval = apic_handle_defconf()) != PSM_SUCCESS)
445                         return (retval);

447                 goto apic_ret;
448         }

450         /* MP Configuration Table */
451         mpct_addr = (uint32_t)(fpsp->mpfps_mpct_paddr);

453         psm_unmap_phys(fptr, mapsize); /* unmap floating ptr struct */

455         /*
456          * Map in enough memory for the MP Configuration Table Header.
457          * Use this table to read the total length of the BIOS data and
458          * map in all the info
459          */
460         /*LINTED: pointer cast may result in improper alignment */
461         hdrp = (struct apic_mp_cnf_hdr *)psm_map_phys(mpct_addr,
462             sizeof (struct apic_mp_cnf_hdr), PROT_READ);
463         if (!hdrp)
464                 goto apic_ret;

466         /* check mp configuration table signature PCMP */
467         if (hdrp->mpcnf_sig != 0x504d4350) {
468                 psm_unmap_phys((caddr_t)hdrp, sizeof (struct apic_mp_cnf_hdr));
469                 goto apic_ret;
470         }
471         mpct_size = (int)hdrp->mpcnf_tbl_length;

473         apic_set_pwroff_method_from_mpcnfhdr(hdrp);

475         psm_unmap_phys((caddr_t)hdrp, sizeof (struct apic_mp_cnf_hdr));

477         if ((retval == PSM_SUCCESS) && !apic_use_acpi_madt_only) {
478                 /* This is an ACPI machine No need for further checks */
479                 goto apic_ret;
480         }

482         /*
483          * Map in the entries for this machine, ie. Processor
484          * Entry Tables, Bus Entry Tables, etc.
485          * They are in fixed order following one another
486          */
487         mpct = psm_map_phys(mpct_addr, mpct_size, PROT_READ);
488         if (!mpct)
489                 goto apic_ret;

491         if (apic_checksum(mpct, mpct_size) != 0)
492                 goto apic_fail1;
```



```
494         /*LINTED: pointer cast may result in improper alignment */
495         hdrp = (struct apic_mp_cnf_hdr *)mpct;
496         apicadr = (uint32_t *)mapin_apic((uint32_t)hdrp->mpcnf_local_apic,
497             APIC_LOCAL_MEMLEN, PROT_READ | PROT_WRITE);
498         if (!apicadr)
499                 goto apic_fail1;

501         /* Parse all information in the tables */
502         bypass_cpu_and_ioapics_in_mptables = (retval == PSM_SUCCESS);
503         if (apic_parse_mpct(mpct, bypass_cpu_and_ioapics_in_mptables) ==
504             PSM_SUCCESS) {
505                 retval = PSM_SUCCESS;
506                 goto apic_ret;
507         }

509 apic_fail1:
510         psm_unmap_phys(mpct, mpct_size);
511         mpct = NULL;

513 apic_ret:
514         if (retval == PSM_SUCCESS) {
515                 extern int apic_ioapic_method_probe();

517                 if ((retval = apic_ioapic_method_probe()) == PSM_SUCCESS)
518                         return (PSM_SUCCESS);
519         }

521         for (i = 0; i < apic_io_max; i++)
522                 mapout_ioapic((caddr_t)apicioadr[i], APIC_IO_MEMLEN);
523         if (apic_cpus) {
524                 kmem_free(apic_cpus, apic_cpus_size);
525                 apic_cpus = NULL;
526         }
527         if (apicadr) {
528                 mapout_apic((caddr_t)apicadr, APIC_LOCAL_MEMLEN);
529                 apicadr = NULL;
530         }
531         if (mpct)
532                 psm_unmap_phys(mpct, mpct_size);

534         return (retval);
535 }
```

*unchanged_portion_omitted_*

```
**********************************************************
   63966 Thu Sep  7 15:25:32 2017
new/usr/src/uts/i86pc/io/mp_platform_misc.c
8626 make pcplusmp and apix warning-free
Reviewed by: Robert Mustacchi <rm@joyent.com>
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright (c) 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright 2017 Joyent, Inc.
 */
/*
 * Copyright (c) 2010, Intel Corporation.
 * All rights reserved.
 */

/*
 * PSMI 1.1 extensions are supported only in 2.6 and later versions.
 * PSMI 1.2 extensions are supported only in 2.7 and later versions.
 * PSMI 1.3 and 1.4 extensions are supported in Solaris 10.
 * PSMI 1.5 extensions are supported in Solaris Nevada.
 * PSMI 1.6 extensions are supported in Solaris Nevada.
 * PSMI 1.7 extensions are supported in Solaris Nevada.
 */
#define PSMI_1_7

#include <sys/processor.h>
#include <sys/time.h>
#include <sys/psm.h>
#include <sys/smp_impldefs.h>
#include <sys/inttypes.h>
#include <sys/cram.h>
#include <sys/acpi/acpi.h>
#include <sys/acpica.h>
#include <sys/psm_common.h>
#include <sys/apic.h>
#include <sys/apic_common.h>
#include <sys/pit.h>
#include <sys/ddi.h>
#include <sys/sunddi.h>
#include <sys/ddi_impldefs.h>
#include <sys/pci.h>
#include <sys/promif.h>
#include <sys/x86_archext.h>
#include <sys/cpc_impl.h>
#include <sys/uadmin.h>
```



```
#include <sys/panic.h>
#include <sys/debug.h>
#include <sys/archsystm.h>
#include <sys/trap.h>
#include <sys/machsystm.h>
#include <sys/cpuvar.h>
#include <sys/rm_platter.h>
#include <sys/privregs.h>
#include <sys/cyclic.h>
#include <sys/note.h>
#include <sys/pci_intr_lib.h>
#include <sys/sunndi.h>
#include <sys/hpet.h>
#include <sys/clock.h>

/*
 * Part of mp_platfrom_common.c that's used only by pcplusmp & xpv_psm
 * but not apix.
 * These functions may be moved to xpv_psm later when apix and pcplusmp
 * are merged together
 */

/*
 *      Local Function Prototypes
 */
static void apic_mark_vector(uchar_t oldvector, uchar_t newvector);
static void apic_xlate_vector_free_timeout_handler(void *arg);
static int apic_check_stuck_interrupt(apic_irq_t *irq_ptr, int old_bind_cpu,
    int new_bind_cpu, int apicindex, int intin_no, int which_irq,
    struct ioapic_reprogram_data *drep);
static int apic_setup_irq_table(dev_info_t *dip, int irqno,
    struct apic_io_intr *intrp, struct intrspec *ispec, iflag_t *intr_flagp,
    int type);
static void apic_try_deferred_reprogram(int ipl, int vect);
static void delete_defer_repro_ent(int which_irq);
static void apic_ioapic_wait_pending_clear(int ioapicindex,
    int intin_no);

extern int apic_acpi_translate_pci_irq(dev_info_t *dip, int busid, int devid,
    int ipin, int *pci_irqp, iflag_t *intr_flagp);
extern int apic_handle_pci_pci_bridge(dev_info_t *idip, int child_devno,
    int child_ipin, struct apic_io_intr **intrp);
extern uchar_t acpi_find_ioapic(int irq);
extern struct apic_io_intr *apic_find_io_intr_w_busid(int irqno, int busid);
extern int apic_find_bus_id(int bustype);
extern int apic_find_intin(uchar_t ioapic, uchar_t intin);
extern void apic_record_rdt_entry(apic_irq_t *irqptr, int irq);

extern  int apic_sci_vect;
extern  iflag_t apic_sci_flags;
/* ACPI HPET interrupt configuration; -1 if HPET not used */
extern  int apic_hpet_vect;
extern  iflag_t apic_hpet_flags;
extern  int     apic_intr_policy;
extern  char *psm_name;

/*
 * number of bits per byte, from <sys/param.h>
 */
#define UCHAR_MAX       UINT8_MAX

/* Max wait time (in repetitions) for flags to clear in an RDT entry. */
extern int apic_max_reps_clear_pending;

/* The irq # is implicit in the array index: */
struct ioapic_reprogram_data apic_reprogram_info[APIC_MAX_VECTOR+1];
```

```
126 /*
127  * APIC_MAX_VECTOR + 1 is the maximum # of IRQs as well. ioapic_reprogram_info
128  * is indexed by IRQ number, NOT by vector number.
129  */

131 extern  int     apic_int_busy_mark;
132 extern  int     apic_int_free_mark;
133 extern  int     apic_diff_for_redistribution;
134 extern  int     apic_sample_factor_redistribution;
135 extern  int     apic_redist_cpu_skip;
136 extern  int     apic_num_imbalance;
137 extern  int     apic_num_rebind;

139 /* timeout for xlate_vector, mark_vector */
140 int     apic_revector_timeout = 16 * 10000; /* 160 millisec */

142 extern int      apic_defconf;
143 extern int      apic_irq_translate;

145 extern int      apic_use_acpi_madt_only;        /* 1=ONLY use MADT from ACPI */

147 extern  uchar_t apic_io_vectbase[MAX_IO_APIC];

149 extern  boolean_t ioapic_mask_workaround[MAX_IO_APIC];

151 /*
152  * First available slot to be used as IRQ index into the apic_irq_table
153  * for those interrupts (like MSI/X) that don't have a physical IRQ.
154  */
155 extern int apic_first_avail_irq;

157 /*
158  * apic_defer_reprogram_lock ensures that only one processor is handling
159  * deferred interrupt programming at *_intr_exit time.
160  */
161 static  lock_t  apic_defer_reprogram_lock;

163 /*
164  * The current number of deferred reprogrammings outstanding
165  */
166 uint_t  apic_reprogram_outstanding = 0;

168 #ifdef DEBUG
169 /*
170  * Counters that keep track of deferred reprogramming stats
171  */
172 uint_t  apic_intr_deferrals = 0;
173 uint_t  apic_intr_deliver_timeouts = 0;
174 uint_t  apic_last_ditch_reprogram_failures = 0;
175 uint_t  apic_deferred_setup_failures = 0;
176 uint_t  apic_defer_repro_total_retries = 0;
177 uint_t  apic_defer_repro_successes = 0;
178 uint_t  apic_deferred_spurious_enters = 0;
179 #endif

181 extern  int     apic_io_max;
182 extern  struct apic_io_intr *apic_io_intrp;

184 uchar_t apic_vector_to_irq[APIC_MAX_VECTOR+1];

186 extern  uint32_t        eisa_level_intr_mask;
187         /* At least MSB will be set if EISA bus */

189 extern  int     apic_pci_bus_total;
190 extern  uchar_t apic_single_pci_busid;
```



```
192 /*
193  * Following declarations are for revectoring; used when ISRs at different
194  * IPLs share an irq.
195  */
196 static  lock_t  apic_revector_lock;
197 int     apic_revector_pending = 0;
198 static  uchar_t *apic_oldvec_to_newvec;
199 static  uchar_t *apic_newvec_to_oldvec;

201 /* ACPI Interrupt Source Override Structure ptr */
202 extern ACPI_MADT_INTERRUPT_OVERRIDE *acpi_isop;
203 extern int acpi_iso_cnt;

205 /*
206  * Auto-configuration routines
207  */

209 /*
210  * Initialise vector->ipl and ipl->pri arrays. level_intr and irqtable
211  * are also set to NULL. vector->irq is set to a value which cannot map
212  * to a real irq to show that it is free.
213  */
214 void
215 apic_init_common(void)
216 {
217         int     i, j, indx;
218         int     *iptr;

220         /*
221          * Initialize apic_ipls from apic_vectortoipl.  This array is
222          * used in apic_intr_enter to determine the IPL to use for the
223          * corresponding vector.  On some systems, due to hardware errata
224          * and interrupt sharing, the IPL may not correspond to the IPL listed
225          * in apic_vectortoipl (see apic_addspl and apic_delspl).
226          */
227         for (i = 0; i < (APIC_AVAIL_VECTOR / APIC_VECTOR_PER_IPL); i++) {
228                 indx = i * APIC_VECTOR_PER_IPL;

230                 for (j = 0; j < APIC_VECTOR_PER_IPL; j++, indx++)
231                         apic_ipls[indx] = apic_vectortoipl[i];
232         }

234         /* cpu 0 is always up (for now) */
235         apic_cpus[0].aci_status = APIC_CPU_ONLINE | APIC_CPU_INTR_ENABLE;

237         iptr = (int *)&apic_irq_table[0];
238         for (i = 0; i <= APIC_MAX_VECTOR; i++) {
239                 apic_level_intr[i] = 0;
240                 *iptr++ = NULL;
241                 apic_vector_to_irq[i] = APIC_RESV_IRQ;

243                 /* These *must* be initted to B_TRUE! */
244                 apic_reprogram_info[i].done = B_TRUE;
245                 apic_reprogram_info[i].irqp = NULL;
246                 apic_reprogram_info[i].tries = 0;
247                 apic_reprogram_info[i].bindcpu = 0;
248         }

250         /*
251          * Allocate a dummy irq table entry for the reserved entry.
252          * This takes care of the race between removing an irq and
253          * clock detecting a CPU in that irq during interrupt load
254          * sampling.
255          */
256         apic_irq_table[APIC_RESV_IRQ] =
257             kmem_zalloc(sizeof (apic_irq_t), KM_SLEEP);
```

```
259         mutex_init(&airq_mutex, NULL, MUTEX_DEFAULT, NULL);
260 }
_____unchanged_portion_omitted_

488 /*
489  * Recompute mask bits for the given interrupt vector.
490  * If there is no interrupt servicing routine for this
491  * vector, this function should disable interrupt vector
492  * from happening at all IPLs. If there are still
493  * handlers using the given vector, this function should
494  * disable the given vector from happening below the lowest
495  * IPL of the remaining hadlers.
496  */
497 /*ARGSUSED*/
498 int
499 apic_delspl_common(int irqno, int ipl, int min_ipl, int max_ipl)
500 {
501         uchar_t vector;
502         uint32_t bind_cpu;
503         int intin, irqindex;
504         int ioapic_ix;
505         apic_irq_t      *irqptr, *preirqptr, *irqheadptr, *irqp;
506         ulong_t iflag;

508         mutex_enter(&airq_mutex);
509         irqindex = IRQINDEX(irqno);
510         irqptr = preirqptr = irqheadptr = apic_irq_table[irqindex];

512         DDI_INTR_IMPLDBG((CE_CONT, "apic_delspl: dip=0x%p type=%d irqno=0x%x "
513             "vector=0x%x\n", (void *)irqptr->airq_dip,
514             irqptr->airq_mps_intr_index, irqno, irqptr->airq_vector));

516         while (irqptr) {
517                 if (VIRTIRQ(irqindex, irqptr->airq_share_id) == irqno)
518                         break;
519                 preirqptr = irqptr;
520                 irqptr = irqptr->airq_next;
521         }
522         ASSERT(irqptr);

524         irqptr->airq_share--;

526         mutex_exit(&airq_mutex);

528         /*
529          * If there are more interrupts at a higher IPL, we don't need
530          * to disable anything.
531          */
532         if (ipl < max_ipl)
533                 return (PSM_SUCCESS);

535         /* return if it is not hardware interrupt */
536         if (irqptr->airq_mps_intr_index == RESERVE_INDEX)
537                 return (PSM_SUCCESS);

539         if (!apic_picinit_called) {
540                 /*
541                  * Clear irq_struct. If two devices shared an intpt
542                  * line & 1 unloaded before picinit, we are hosed. But, then
543                  * we hope the machine survive.
544                  */
545                 irqptr->airq_mps_intr_index = FREE_INDEX;
546                 irqptr->airq_temp_cpu = IRQ_UNINIT;
547                 apic_free_vector(irqptr->airq_vector);
548                 return (PSM_SUCCESS);
```



```
549         }
550         /*
551          * Downgrade vector to new max_ipl if needed. If we cannot allocate,
552          * use old IPL. Not very elegant, but it should work.
553          */
554         if ((irqptr->airq_ipl != max_ipl) && (max_ipl != PSM_INVALID_IPL) &&
555             !ioapic_mask_workaround[irqptr->airq_ioapicindex]) {
556                 apic_irq_t      *irqp;
557                 if ((vector = apic_allocate_vector(max_ipl, irqno, 1))) {
558                         apic_mark_vector(irqheadptr->airq_vector, vector);
559                         irqp = irqheadptr;
560                         while (irqp) {
561                                 irqp->airq_vector = vector;
562                                 irqp->airq_ipl = (uchar_t)max_ipl;
563                                 if (irqp->airq_temp_cpu != IRQ_UNINIT) {
564                                         apic_record_rdt_entry(irqp, irqindex);

566                                         iflag = intr_clear();
567                                         lock_set(&apic_ioapic_lock);

569                                         (void) apic_setup_io_intr(irqp,
570                                             irqindex, B_FALSE);

572                                         lock_clear(&apic_ioapic_lock);
573                                         intr_restore(iflag);
574                                 }
575                                 irqp = irqp->airq_next;
576                         }
577                 }

579         } else if (irqptr->airq_ipl != max_ipl &&
580             max_ipl != PSM_INVALID_IPL &&
581             ioapic_mask_workaround[irqptr->airq_ioapicindex]) {

583         /*
584          * We cannot downgrade the IPL of the vector below the vector's
585          * hardware priority. If we did, it would be possible for a
586          * higher-priority hardware vector to interrupt a CPU running at an IPL
587          * lower than the hardware priority of the interrupting vector (but
588          * higher than the soft IPL of this IRQ). When this happens, we would
589          * then try to drop the IPL BELOW what it was (effectively dropping
590          * below base_spl) which would be potentially catastrophic.
591          *
592          * (e.g. Suppose the hardware vector associated with this IRQ is 0x40
593          * (hardware IPL of 4).  Further assume that the old IPL of this IRQ
594          * was 4, but the new IPL is 1.  If we forced vector 0x40 to result in
595          * an IPL of 1, it would be possible for the processor to be executing
596          * at IPL 3 and for an interrupt to come in on vector 0x40, interrupting
597          * the currently-executing ISR.  When apic_intr_enter consults
598          * apic_irqs[], it will return 1, bringing the IPL of the CPU down to 1
599          * so even though the processor was running at IPL 4, an IPL 1
600          * interrupt will have interrupted it, which must not happen)).
601          *
602          * Effectively, this means that the hardware priority corresponding to
603          * the IRQ's IPL (in apic_ipls[]) cannot be lower than the vector's
604          * hardware priority.
605          *
606          * (In the above example, then, after removal of the IPL 4 device's
607          * interrupt handler, the new IPL will continue to be 4 because the
608          * hardware priority that IPL 1 implies is lower than the hardware
609          * priority of the vector used.)
610          */
611                 /* apic_ipls is indexed by vector, starting at APIC_BASE_VECT */
612                 const int apic_ipls_index = irqptr->airq_vector -
613                     APIC_BASE_VECT;
614                 const int vect_inherent_hwpri = irqptr->airq_vector >>
```

```
615                     APIC_IPL_SHIFT;

617             /*
618              * If there are still devices using this IRQ, determine the
619              * new ipl to use.
620              */
621             if (irqptr->airq_share) {
622                     int vect_desired_hwpri, hwpri;

624                     ASSERT(max_ipl < MAXIPL);
625                     vect_desired_hwpri = apic_ipltopri[max_ipl] >>
626                         APIC_IPL_SHIFT;

628                     /*
629                      * If the desired IPL's hardware priority is lower
630                      * than that of the vector, use the hardware priority
631                      * of the vector to determine the new IPL.
632                      */
633                     hwpri = (vect_desired_hwpri < vect_inherent_hwpri) ?
634                         vect_inherent_hwpri : vect_desired_hwpri;

636                     /*
637                      * Now, to get the right index for apic_vectortoipl,
638                      * we need to subtract APIC_BASE_VECT from the
639                      * hardware-vector-equivalent (in hwpri).  Since hwpri
640                      * is already shifted, we shift APIC_BASE_VECT before
641                      * doing the subtraction.
642                      */
643                     hwpri -= (APIC_BASE_VECT >> APIC_IPL_SHIFT);

645                     ASSERT(hwpri >= 0);
646                     ASSERT(hwpri < MAXIPL);
647                     max_ipl = apic_vectortoipl[hwpri];
648                     apic_ipls[apic_ipls_index] = (uchar_t)max_ipl;
647                     apic_ipls[apic_ipls_index] = max_ipl;

650                     irqp = irqheadptr;
651                     while (irqp) {
652                             irqp->airq_ipl = (uchar_t)max_ipl;
653                             irqp = irqp->airq_next;
654                     }
655             } else {
656                     /*
657                      * No more devices on this IRQ, so reset this vector's
658                      * element in apic_ipls to the original IPL for this
659                      * vector
660                      */
661                     apic_ipls[apic_ipls_index] =
662                         apic_vectortoipl[vect_inherent_hwpri];
663             }
664         }

666         /*
667          * If there are still active interrupts, we are done.
668          */
669         if (irqptr->airq_share)
670                 return (PSM_SUCCESS);

672         iflag = intr_clear();
673         lock_set(&apic_ioapic_lock);

675         if (irqptr->airq_mps_intr_index == MSI_INDEX) {
676                 /*
677                  * Disable the MSI vector
678                  * Make sure we only disable on the last
679                  * of the multi-MSI support
```



```
680                  */
681                 if (i_ddi_intr_get_current_nenables(irqptr->airq_dip) == 1) {
682                         apic_pci_msi_disable_mode(irqptr->airq_dip,
683                             DDI_INTR_TYPE_MSI);
684                 }
685         } else if (irqptr->airq_mps_intr_index == MSIX_INDEX) {
686                 /*
687                  * Disable the MSI-X vector
688                  * needs to clear its mask and addr/data for each MSI-X
689                  */
690                 apic_pci_msi_unconfigure(irqptr->airq_dip, DDI_INTR_TYPE_MSIX,
691                     irqptr->airq_origirq);
692                 /*
693                  * Make sure we only disable on the last MSI-X
694                  */
695                 if (i_ddi_intr_get_current_nenables(irqptr->airq_dip) == 1) {
696                         apic_pci_msi_disable_mode(irqptr->airq_dip,
697                             DDI_INTR_TYPE_MSIX);
698                 }
699         } else {
700                 /*
701                  * The assumption here is that this is safe, even for
702                  * systems with IOAPICs that suffer from the hardware
703                  * erratum because all devices have been quiesced before
704                  * they unregister their interrupt handlers.  If that
705                  * assumption turns out to be false, this mask operation
706                  * can induce the same erratum result we're trying to
707                  * avoid.
708                  */
709                 ioapic_ix = irqptr->airq_ioapicindex;
710                 intin = irqptr->airq_intin_no;
711                 ioapic_write(ioapic_ix, APIC_RDT_CMD + 2 * intin, AV_MASK);
712         }

714         apic_vt_ops->apic_intrmap_free_entry(&irqptr->airq_intrmap_private);

716         /*
717          * This irq entry is the only one in the chain.
718          */
719         if (irqheadptr->airq_next == NULL) {
720                 ASSERT(irqheadptr == irqptr);
721                 bind_cpu = irqptr->airq_temp_cpu;
722                 if (((uint32_t)bind_cpu != IRQ_UNBOUND) &&
723                     ((uint32_t)bind_cpu != IRQ_UNINIT)) {
724                         ASSERT(apic_cpu_in_range(bind_cpu));
725                         if (bind_cpu & IRQ_USER_BOUND) {
726                                 /* If hardbound, temp_cpu == cpu */
727                                 bind_cpu &= ~IRQ_USER_BOUND;
728                                 apic_cpus[bind_cpu].aci_bound--;
729                         } else
730                                 apic_cpus[bind_cpu].aci_temp_bound--;
731                 }
732                 irqptr->airq_temp_cpu = IRQ_UNINIT;
733                 irqptr->airq_mps_intr_index = FREE_INDEX;
734                 lock_clear(&apic_ioapic_lock);
735                 intr_restore(iflag);
736                 apic_free_vector(irqptr->airq_vector);
737                 return (PSM_SUCCESS);
738         }

740         /*
741          * If we get here, we are sharing the vector and there are more than
742          * one active irq entries in the chain.
743          */
744         lock_clear(&apic_ioapic_lock);
745         intr_restore(iflag);
```

```
 747         mutex_enter(&airq_mutex);
 748         /* Remove the irq entry from the chain */
 749         if (irqptr == irqheadptr) { /* The irq entry is at the head */
 750                 apic_irq_table[irqindex] = irqptr->airq_next;
 751         } else {
 752                 preirqptr->airq_next = irqptr->airq_next;
 753         }
 754         /* Free the irq entry */
 755         kmem_free(irqptr, sizeof (apic_irq_t));
 756         mutex_exit(&airq_mutex);

 758         return (PSM_SUCCESS);
 759 }
_____unchanged_portion_omitted_

1042 /*
1043  * Allocate/Initialize the apic_irq_table[] entry for given irqno. If the entry
1044  * is used already, we will try to allocate a new irqno.
1045  *
1046  * Return value:
1047  *      Success: irqno
1048  *      Failure: -1
1049  */
1050 static int
1051 apic_setup_irq_table(dev_info_t *dip, int irqno, struct apic_io_intr *intrp,
1052     struct intrspec *ispec, iflag_t *intr_flagp, int type)
1053 {
1054         int origirq;
1055         uchar_t ipl;
1053         int origirq = ispec->intrspec_vec;
1054         uchar_t ipl = ispec->intrspec_pri;
1056         int     newirq, intr_index;
1057         uchar_t ipin, ioapic, ioapicindex, vector;
1058         apic_irq_t *irqptr;
1059         major_t major;
1060         dev_info_t      *sdip;

1062         ASSERT(ispec != NULL);

1064         origirq = ispec->intrspec_vec;
1065         ipl = ispec->intrspec_pri;

1067         DDI_INTR_IMPLDBG((CE_CONT, "apic_setup_irq_table: dip=0x%p type=%d "
1068             "irqno=0x%x origirq=0x%x\n", (void *)dip, type, irqno, origirq));

1064         ASSERT(ispec != NULL);

1070         major =  (dip != NULL) ? ddi_driver_major(dip) : 0;

1072         if (DDI_INTR_IS_MSI_OR_MSIX(type)) {
1073                 /* MSI/X doesn't need to setup ioapic stuffs */
1074                 ioapicindex = 0xff;
1075                 ioapic = 0xff;
1076                 ipin = (uchar_t)0xff;
1077                 intr_index = (type == DDI_INTR_TYPE_MSI) ? MSI_INDEX :
1078                     MSIX_INDEX;
1079                 mutex_enter(&airq_mutex);
1080                 if ((irqno = apic_allocate_irq(apic_first_avail_irq)) == -1) {
1081                         mutex_exit(&airq_mutex);
1082                         /* need an irq for MSI/X to index into autovect[] */
1083                         cmn_err(CE_WARN, "No interrupt irq: %s instance %d",
1084                             ddi_get_name(dip), ddi_get_instance(dip));
1085                         return (-1);
1086                 }
1087                 mutex_exit(&airq_mutex);
```



```
1089         } else if (intrp != NULL) {
1090                 intr_index = (int)(intrp - apic_io_intrp);
1091                 ioapic = intrp->intr_destid;
1092                 ipin = intrp->intr_destintin;
1093                 /* Find ioapicindex. If destid was ALL, we will exit with 0. */
1094                 for (ioapicindex = apic_io_max - 1; ioapicindex; ioapicindex--)
1095                         if (apic_io_id[ioapicindex] == ioapic)
1096                                 break;
1097                 ASSERT((ioapic == apic_io_id[ioapicindex]) ||
1098                     (ioapic == INTR_ALL_APIC));

1100                 /* check whether this intin# has been used by another irqno */
1101                 if ((newirq = apic_find_intin(ioapicindex, ipin)) != -1) {
1102                         return (newirq);
1103                 }

1105         } else if (intr_flagp != NULL) {
1106                 /* ACPI case */
1107                 intr_index = ACPI_INDEX;
1108                 ioapicindex = acpi_find_ioapic(irqno);
1109                 ASSERT(ioapicindex != 0xFF);
1110                 ioapic = apic_io_id[ioapicindex];
1111                 ipin = irqno - apic_io_vectbase[ioapicindex];
1112                 if (apic_irq_table[irqno] &&
1113                     apic_irq_table[irqno]->airq_mps_intr_index == ACPI_INDEX) {
1114                         ASSERT(apic_irq_table[irqno]->airq_intin_no == ipin &&
1115                             apic_irq_table[irqno]->airq_ioapicindex ==
1116                             ioapicindex);
1117                         return (irqno);
1118                 }

1120         } else {
1121                 /* default configuration */
1122                 ioapicindex = 0;
1123                 ioapic = apic_io_id[ioapicindex];
1124                 ipin = (uchar_t)irqno;
1125                 intr_index = DEFAULT_INDEX;
1126         }

1128         if ((vector = apic_allocate_vector(ipl, irqno, 0)) == 0) {
1124         if (ispec == NULL) {
1125                 APIC_VERBOSE_IOAPIC((CE_WARN, "No intrspec for irqno = %x\n",
1126                     irqno));
1127         } else if ((vector = apic_allocate_vector(ipl, irqno, 0)) == 0) {
1129                 if ((newirq = apic_share_vector(irqno, intr_flagp, intr_index,
1130                     ipl, ioapicindex, ipin, &irqptr)) != -1) {
1131                         irqptr->airq_ipl = ipl;
1132                         irqptr->airq_origirq = (uchar_t)origirq;
1133                         irqptr->airq_dip = dip;
1134                         irqptr->airq_major = major;
1135                         sdip = apic_irq_table[IRQINDEX(newirq)]->airq_dip;
1136                         /* This is OK to do really */
1137                         if (sdip == NULL) {
1138                                 cmn_err(CE_WARN, "Sharing vectors: %s"
1139                                     " instance %d and SCI",
1140                                     ddi_get_name(dip), ddi_get_instance(dip));
1141                         } else {
1142                                 cmn_err(CE_WARN, "Sharing vectors: %s"
1143                                     " instance %d and %s instance %d",
1144                                     ddi_get_name(sdip), ddi_get_instance(sdip),
1145                                     ddi_get_name(dip), ddi_get_instance(dip));
1146                         }
1147                         return (newirq);
1148                 }
1149                 /* try high priority allocation now  that share has failed */
```

```
1150                 if ((vector = apic_allocate_vector(ipl, irqno, 1)) == 0) {
1151                         cmn_err(CE_WARN, "No interrupt vector: %s instance %d",
1152                             ddi_get_name(dip), ddi_get_instance(dip));
1153                         return (-1);
1154                 }
1155         }

1157         mutex_enter(&airq_mutex);
1158         if (apic_irq_table[irqno] == NULL) {
1159                 irqptr = kmem_zalloc(sizeof (apic_irq_t), KM_SLEEP);
1160                 irqptr->airq_temp_cpu = IRQ_UNINIT;
1161                 apic_irq_table[irqno] = irqptr;
1162         } else {
1163                 irqptr = apic_irq_table[irqno];
1164                 if (irqptr->airq_mps_intr_index != FREE_INDEX) {
1165                         /*
1166                          * The slot is used by another irqno, so allocate
1167                          * a free irqno for this interrupt
1168                          */
1169                         newirq = apic_allocate_irq(apic_first_avail_irq);
1170                         if (newirq == -1) {
1171                                 mutex_exit(&airq_mutex);
1172                                 return (-1);
1173                         }
1174                         irqno = newirq;
1175                         irqptr = apic_irq_table[irqno];
1176                         if (irqptr == NULL) {
1177                                 irqptr = kmem_zalloc(sizeof (apic_irq_t),
1178                                     KM_SLEEP);
1179                                 irqptr->airq_temp_cpu = IRQ_UNINIT;
1180                                 apic_irq_table[irqno] = irqptr;
1181                         }
1182                         vector = apic_modify_vector(vector, newirq);
1183                 }
1184         }
1185         apic_max_device_irq = max(irqno, apic_max_device_irq);
1186         apic_min_device_irq = min(irqno, apic_min_device_irq);
1187         mutex_exit(&airq_mutex);
1188         irqptr->airq_ioapicindex = ioapicindex;
1189         irqptr->airq_intin_no = ipin;
1190         irqptr->airq_ipl = ipl;
1191         irqptr->airq_vector = vector;
1192         irqptr->airq_origirq = (uchar_t)origirq;
1193         irqptr->airq_share_id = 0;
1194         irqptr->airq_mps_intr_index = (short)intr_index;
1195         irqptr->airq_dip = dip;
1196         irqptr->airq_major = major;
1197         irqptr->airq_cpu = apic_bind_intr(dip, irqno, ioapic, ipin);
1198         if (intr_flagp)
1199                 irqptr->airq_iflag = *intr_flagp;

1201         if (!DDI_INTR_IS_MSI_OR_MSIX(type)) {
1202                 /* setup I/O APIC entry for non-MSI/X interrupts */
1203                 apic_record_rdt_entry(irqptr, irqno);
1204         }
1205         return (irqno);
1206 }

1208 /*
1209  * return the cpu to which this intr should be bound.
1210  * Check properties or any other mechanism to see if user wants it
1211  * bound to a specific CPU. If so, return the cpu id with high bit set.
1212  * If not, use the policy to choose a cpu and return the id.
1213  */
1214 uint32_t
1215 apic_bind_intr(dev_info_t *dip, int irq, uchar_t ioapicid, uchar_t intin)
```



```
1216 {
1217         int     instance, instno, prop_len, bind_cpu, count;
1218         uint_t  i, rc;
1219         uint32_t cpu;
1220         major_t major;
1221         char    *name, *drv_name, *prop_val, *cptr;
1222         char    prop_name[32];
1223         ulong_t iflag;


1226         if (apic_intr_policy == INTR_LOWEST_PRIORITY)
1227                 return (IRQ_UNBOUND);

1229         if (apic_nproc == 1)
1230                 return (0);

1232         if (dip == NULL) {
1233                 iflag = intr_clear();
1234                 lock_set(&apic_ioapic_lock);
1235                 bind_cpu = apic_get_next_bind_cpu();
1236                 lock_clear(&apic_ioapic_lock);
1237                 intr_restore(iflag);

1239                 cmn_err(CE_CONT, "!%s: irq 0x%x "
1240                     "vector 0x%x ioapic 0x%x intin 0x%x is bound to cpu %d\n",
1241                     psm_name, irq, apic_irq_table[irq]->airq_vector, ioapicid,
1242                     intin, bind_cpu & ~IRQ_USER_BOUND);

1244                 return ((uint32_t)bind_cpu);
1245         }

1231         drv_name = NULL;
1232         rc = DDI_PROP_NOT_FOUND;
1233         major = (major_t)-1;
1234         if (dip != NULL) {
1247                 name = ddi_get_name(dip);
1248                 major = ddi_name_to_major(name);
1249                 drv_name = ddi_major_to_name(major);
1250                 instance = ddi_get_instance(dip);
1251                 if (apic_intr_policy == INTR_ROUND_ROBIN_WITH_AFFINITY) {
1252                         i = apic_min_device_irq;
1253                         for (; i <= apic_max_device_irq; i++) {

1254                                 if ((i == irq) || (apic_irq_table[i] == NULL) ||
1255                                     (apic_irq_table[i]->airq_mps_intr_index
1256                                     == FREE_INDEX))
1257                                         continue;

1259                                 if ((apic_irq_table[i]->airq_major == major) &&
1260                                     (!(apic_irq_table[i]->airq_cpu & IRQ_USER_BOUND))) {
1249                                         (!(apic_irq_table[i]->airq_cpu &
1250                                         IRQ_USER_BOUND))) {

1261                                         cpu = apic_irq_table[i]->airq_cpu;

1263                                         cmn_err(CE_CONT,
1264                                             "!%s: %s (%s) instance #%d "
1265                                             "irq 0x%x vector 0x%x ioapic 0x%x "
1266                                             "intin 0x%x is bound to cpu %d\n",
1267                                             psm_name,
1268                                             name, drv_name, instance, irq,
1269                                             apic_irq_table[irq]->airq_vector,
1270                                             ioapicid, intin, cpu);
1271                                         return (cpu);
1272                                 }
1273                         }
```

```
1274         }
1275         /*
1276          * search for "drvname"_intpt_bind_cpus property first, the
1277          * syntax of the property should be "a[,b,c,...]" where
1278          * instance 0 binds to cpu a, instance 1 binds to cpu b,
1279          * instance 3 binds to cpu c...
1280          * ddi_getlongprop() will search /option first, then /
1281          * if "drvname"_intpt_bind_cpus doesn't exist, then find
1282          * intpt_bind_cpus property.  The syntax is the same, and
1283          * it applies to all the devices if its "drvname" specific
1284          * property doesn't exist
1285          */
1286         (void) strcpy(prop_name, drv_name);
1287         (void) strcat(prop_name, "_intpt_bind_cpus");
1288         rc = ddi_getlongprop(DDI_DEV_T_ANY, dip, 0, prop_name,
1289             (caddr_t)&prop_val, &prop_len);
1290         if (rc != DDI_PROP_SUCCESS) {
1291                 rc = ddi_getlongprop(DDI_DEV_T_ANY, dip, 0,
1292                     "intpt_bind_cpus", (caddr_t)&prop_val, &prop_len);
1293         }
1285         }
1294         if (rc == DDI_PROP_SUCCESS) {
1295                 for (i = count = 0; i < (prop_len - 1); i++)
1296                         if (prop_val[i] == ',')
1297                                 count++;
1298                 if (prop_val[i-1] != ',')
1299                         count++;
1300                 /*
1301                  * if somehow the binding instances defined in the
1302                  * property are not enough for this instno., then
1303                  * reuse the pattern for the next instance until
1304                  * it reaches the requested instno
1305                  */
1306                 instno = instance % count;
1307                 i = 0;
1308                 cptr = prop_val;
1309                 while (i < instno)
1310                         if (*cptr++ == ',')
1311                                 i++;
1312                 bind_cpu = stoi(&cptr);
1313                 kmem_free(prop_val, prop_len);
1314                 /* if specific CPU is bogus, then default to next cpu */
1315                 if (!apic_cpu_in_range(bind_cpu)) {
1316                         cmn_err(CE_WARN, "%s: %s=%s: CPU %d not present",
1317                             psm_name, prop_name, prop_val, bind_cpu);
1318                         rc = DDI_PROP_NOT_FOUND;
1319                 } else {
1320                         /* indicate that we are bound at user request */
1321                         bind_cpu |= IRQ_USER_BOUND;
1322                 }
1323                 /*
1324                  * no need to check apic_cpus[].aci_status, if specific CPU is
1325                  * not up, then post_cpu_start will handle it.
1326                  */
1327         }

1329         if (rc != DDI_PROP_SUCCESS) {
1330                 iflag = intr_clear();
1331                 lock_set(&apic_ioapic_lock);
1332                 bind_cpu = apic_get_next_bind_cpu();
1333                 lock_clear(&apic_ioapic_lock);
1334                 intr_restore(iflag);
1335         }

1328         if (drv_name != NULL)
1337                 cmn_err(CE_CONT, "!%s: %s (%s) instance %d irq 0x%x "
```



```
1338                     "vector 0x%x ioapic 0x%x intin 0x%x is bound to cpu %d\n",
1339                     psm_name, name, drv_name, instance, irq,
1340                     apic_irq_table[irq]->airq_vector, ioapicid, intin,
1341                     bind_cpu & ~IRQ_USER_BOUND);
1334         else
1335                 cmn_err(CE_CONT, "!%s: irq 0x%x "
1336                     "vector 0x%x ioapic 0x%x intin 0x%x is bound to cpu %d\n",
1337                     psm_name, irq, apic_irq_table[irq]->airq_vector, ioapicid,
1338                     intin, bind_cpu & ~IRQ_USER_BOUND);

1343         return ((uint32_t)bind_cpu);
1344 }
```

**_____unchanged_portion_omitted_**

```
**********************************************************
   34798 Thu Sep  7 15:25:32 2017
new/usr/src/uts/i86pc/io/pcplusmp/apic.c
8626 make pcplusmp and apix warning-free
Reviewed by: Robert Mustacchi <rm@joyent.com>
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright (c) 1993, 2010, Oracle and/or its affiliates. All rights reserved.
 */
/*
 * Copyright (c) 2010, Intel Corporation.
 * All rights reserved.
 */
/*
 * Copyright (c) 2017, Joyent, Inc.  All rights reserved.
 */

/*
 * To understand how the pcplusmp module interacts with the interrupt subsystem
 * read the theory statement in uts/i86pc/os/intr.c.
 */

/*
 * PSMI 1.1 extensions are supported only in 2.6 and later versions.
 * PSMI 1.2 extensions are supported only in 2.7 and later versions.
 * PSMI 1.3 and 1.4 extensions are supported in Solaris 10.
 * PSMI 1.5 extensions are supported in Solaris Nevada.
 * PSMI 1.6 extensions are supported in Solaris Nevada.
 * PSMI 1.7 extensions are supported in Solaris Nevada.
 */
#define PSMI_1_7

#include <sys/processor.h>
#include <sys/time.h>
#include <sys/psm.h>
#include <sys/smp_impldefs.h>
#include <sys/cram.h>
#include <sys/acpi/acpi.h>
#include <sys/acpica.h>
#include <sys/psm_common.h>
#include <sys/apic.h>
#include <sys/pit.h>
#include <sys/ddi.h>
#include <sys/sunddi.h>
```



```
#include <sys/ddi_impldefs.h>
#include <sys/pci.h>
#include <sys/promif.h>
#include <sys/x86_archext.h>
#include <sys/cpc_impl.h>
#include <sys/uadmin.h>
#include <sys/panic.h>
#include <sys/debug.h>
#include <sys/archsystm.h>
#include <sys/trap.h>
#include <sys/machsystm.h>
#include <sys/sysmacros.h>
#include <sys/cpuvar.h>
#include <sys/rm_platter.h>
#include <sys/privregs.h>
#include <sys/note.h>
#include <sys/pci_intr_lib.h>
#include <sys/spl.h>
#include <sys/clock.h>
#include <sys/cyclic.h>
#include <sys/dditypes.h>
#include <sys/sunddi.h>
#include <sys/x_call.h>
#include <sys/reboot.h>
#include <sys/hpet.h>
#include <sys/apic_common.h>
#include <sys/apic_timer.h>

/*
 *      Local Function Prototypes
 */
static void apic_init_intr(void);

/*
 *      standard MP entries
 */
static int      apic_probe(void);
static int      apic_getclkirq(int ipl);
static void     apic_init(void);
static void     apic_picinit(void);
static int      apic_post_cpu_start(void);
static int      apic_intr_enter(int ipl, int *vect);
static void     apic_setspl(int ipl);
static void     x2apic_setspl(int ipl);
static int      apic_addspl(int ipl, int vector, int min_ipl, int max_ipl);
static int      apic_delspl(int ipl, int vector, int min_ipl, int max_ipl);
static int      apic_disable_intr(processorid_t cpun);
static void     apic_enable_intr(processorid_t cpun);
static int              apic_get_ipivect(int ipl, int type);
static void     apic_post_cyclic_setup(void *arg);

#define UCHAR_MAX       UINT8_MAX

/*
 * The following vector assignments influence the value of ipltopri and
 * vectortoipl. Note that vectors 0 - 0x1f are not used. We can program
 * idle to 0 and IPL 0 to 0xf to differentiate idle in case
 * we care to do so in future. Note some IPLs which are rarely used
 * will share the vector ranges and heavily used IPLs (5 and 6) have
 * a wide range.
 *
 * This array is used to initialize apic_ipls[] (in apic_init()).
 *
 *      IPL             Vector range.           as passed to intr_enter
 *      0               none.
 *      1,2,3           0x20-0x2f               0x0-0xf
```

```
 126  *      4               0x30-0x3f               0x10-0x1f
 127  *      5               0x40-0x5f               0x20-0x3f
 128  *      6               0x60-0x7f               0x40-0x5f
 129  *      7,8,9           0x80-0x8f               0x60-0x6f
 130  *      10              0x90-0x9f               0x70-0x7f
 131  *      11              0xa0-0xaf               0x80-0x8f
 132  *      ...             ...
 133  *      15              0xe0-0xef               0xc0-0xcf
 134  *      15              0xf0-0xff               0xd0-0xdf
 135  */
 136 uchar_t apic_vectortoipl[APIC_AVAIL_VECTOR / APIC_VECTOR_PER_IPL] = {
 137         3, 4, 5, 5, 6, 6, 9, 10, 11, 12, 13, 14, 15, 15
 138 };
_____unchanged_portion_omitted_

1064 /*
1065  * This function allocates "count" MSI vector(s) for the given "dip/pri/type"
1066  */
1067 int
1068 apic_alloc_msi_vectors(dev_info_t *dip, int inum, int count, int pri,
1069     int behavior)
1070 {
1071         int     rcount, i;
1072         uchar_t start, irqno;
1073         uint32_t cpu = 0;
1071         uint32_t cpu;
1074         major_t major;
1075         apic_irq_t      *irqptr;

1077         DDI_INTR_IMPLDBG((CE_CONT, "apic_alloc_msi_vectors: dip=0x%p "
1078             "inum=0x%x  pri=0x%x count=0x%x behavior=%d\n",
1079             (void *)dip, inum, pri, count, behavior));

1081         if (count > 1) {
1082                 if (behavior == DDI_INTR_ALLOC_STRICT &&
1083                     apic_multi_msi_enable == 0)
1084                         return (0);
1085                 if (apic_multi_msi_enable == 0)
1086                         count = 1;
1087         }

1089         if ((rcount = apic_navail_vector(dip, pri)) > count)
1090                 rcount = count;
1091         else if (rcount == 0 || (rcount < count &&
1092             behavior == DDI_INTR_ALLOC_STRICT))
1093                 return (0);

1095         /* if not ISP2, then round it down */
1096         if (!ISP2(rcount))
1097                 rcount = 1 << (highbit(rcount) - 1);

1099         mutex_enter(&airq_mutex);

1101         for (start = 0; rcount > 0; rcount >>= 1) {
1102                 if ((start = apic_find_multi_vectors(pri, rcount)) != 0 ||
1103                     behavior == DDI_INTR_ALLOC_STRICT)
1104                         break;
1105         }

1107         if (start == 0) {
1108                 /* no vector available */
1109                 mutex_exit(&airq_mutex);
1110                 return (0);
1111         }

1113         if (apic_check_free_irqs(rcount) == PSM_FAILURE) {
```



```
1114                 /* not enough free irq slots available */
1115                 mutex_exit(&airq_mutex);
1116                 return (0);
1117         }

1119         major = (dip != NULL) ? ddi_driver_major(dip) : 0;
1120         for (i = 0; i < rcount; i++) {
1121                 if ((irqno = apic_allocate_irq(apic_first_avail_irq)) ==
1122                     (uchar_t)-1) {
1123                         /*
1124                          * shouldn't happen because of the
1125                          * apic_check_free_irqs() check earlier
1126                          */
1127                         mutex_exit(&airq_mutex);
1128                         DDI_INTR_IMPLDBG((CE_CONT, "apic_alloc_msi_vectors: "
1129                             "apic_allocate_irq failed\n"));
1130                         return (i);
1131                 }
1132                 apic_max_device_irq = max(irqno, apic_max_device_irq);
1133                 apic_min_device_irq = min(irqno, apic_min_device_irq);
1134                 irqptr = apic_irq_table[irqno];
1135 #ifdef  DEBUG
1136                 if (apic_vector_to_irq[start + i] != APIC_RESV_IRQ)
1137                         DDI_INTR_IMPLDBG((CE_CONT, "apic_alloc_msi_vectors: "
1138                             "apic_vector_to_irq is not APIC_RESV_IRQ\n"));
1139 #endif
1140                 apic_vector_to_irq[start + i] = (uchar_t)irqno;

1142                 irqptr->airq_vector = (uchar_t)(start + i);
1143                 irqptr->airq_ioapicindex = (uchar_t)inum;       /* start */
1144                 irqptr->airq_intin_no = (uchar_t)rcount;
1145                 ASSERT(pri >= 0 && pri <= UCHAR_MAX);
1146                 irqptr->airq_ipl = (uchar_t)pri;
1143                 irqptr->airq_ipl = pri;
1147                 irqptr->airq_vector = start + i;
1148                 irqptr->airq_origirq = (uchar_t)(inum + i);
1149                 irqptr->airq_share_id = 0;
1150                 irqptr->airq_mps_intr_index = MSI_INDEX;
1151                 irqptr->airq_dip = dip;
1152                 irqptr->airq_major = major;
1153                 if (i == 0) /* they all bound to the same cpu */
1154                         cpu = irqptr->airq_cpu = apic_bind_intr(dip, irqno,
1155                             0xff, 0xff);
1156                 else
1157                         irqptr->airq_cpu = cpu;
1158                 DDI_INTR_IMPLDBG((CE_CONT, "apic_alloc_msi_vectors: irq=0x%x "
1159                     "dip=0x%p vector=0x%x origirq=0x%x pri=0x%x\n", irqno,
1160                     (void *)irqptr->airq_dip, irqptr->airq_vector,
1161                     irqptr->airq_origirq, pri));
1162         }
1163         mutex_exit(&airq_mutex);
1164         return (rcount);
1165 }

1167 /*
1168  * This function allocates "count" MSI-X vector(s) for the given "dip/pri/type"
1169  */
1170 int
1171 apic_alloc_msix_vectors(dev_info_t *dip, int inum, int count, int pri,
1172     int behavior)
1173 {
1174         int     rcount, i;
1175         major_t major;

1177         mutex_enter(&airq_mutex);
```

```
1179         if ((rcount = apic_navail_vector(dip, pri)) > count)
1180                 rcount = count;
1181         else if (rcount == 0 || (rcount < count &&
1182             behavior == DDI_INTR_ALLOC_STRICT)) {
1183                 rcount = 0;
1184                 goto out;
1185         }

1187         if (apic_check_free_irqs(rcount) == PSM_FAILURE) {
1188                 /* not enough free irq slots available */
1189                 rcount = 0;
1190                 goto out;
1191         }

1193         major = (dip != NULL) ? ddi_driver_major(dip) : 0;
1194         for (i = 0; i < rcount; i++) {
1195                 uchar_t vector, irqno;
1196                 apic_irq_t      *irqptr;

1198                 if ((irqno = apic_allocate_irq(apic_first_avail_irq)) ==
1199                     (uchar_t)-1) {
1200                         /*
1201                          * shouldn't happen because of the
1202                          * apic_check_free_irqs() check earlier
1203                          */
1204                         DDI_INTR_IMPLDBG((CE_CONT, "apic_alloc_msix_vectors: "
1205                             "apic_allocate_irq failed\n"));
1206                         rcount = i;
1207                         goto out;
1208                 }
1209                 if ((vector = apic_allocate_vector(pri, irqno, 1)) == 0) {
1210                         /*
1211                          * shouldn't happen because of the
1212                          * apic_navail_vector() call earlier
1213                          */
1214                         DDI_INTR_IMPLDBG((CE_CONT, "apic_alloc_msix_vectors: "
1215                             "apic_allocate_vector failed\n"));
1216                         rcount = i;
1217                         goto out;
1218                 }
1219                 apic_max_device_irq = max(irqno, apic_max_device_irq);
1220                 apic_min_device_irq = min(irqno, apic_min_device_irq);
1221                 irqptr = apic_irq_table[irqno];
1222                 irqptr->airq_vector = (uchar_t)vector;
1223                 ASSERT(pri >= 0 && pri <= UCHAR_MAX);
1224                 irqptr->airq_ipl = (uchar_t)pri;
1220                 irqptr->airq_ipl = pri;
1225                 irqptr->airq_origirq = (uchar_t)(inum + i);
1226                 irqptr->airq_share_id = 0;
1227                 irqptr->airq_mps_intr_index = MSIX_INDEX;
1228                 irqptr->airq_dip = dip;
1229                 irqptr->airq_major = major;
1230                 irqptr->airq_cpu = apic_bind_intr(dip, irqno, 0xff, 0xff);
1231         }
1232 out:
1233         mutex_exit(&airq_mutex);
1234         return (rcount);
1235 }

1237 /*
1238  * Allocate a free vector for irq at ipl. Takes care of merging of multiple
1239  * IPLs into a single APIC level as well as stretching some IPLs onto multiple
1240  * levels. APIC_HI_PRI_VECTS interrupts are reserved for high priority
1241  * requests and allocated only when pri is set.
1242  */
1243 uchar_t
```



```
1244 apic_allocate_vector(int ipl, int irq, int pri)
1245 {
1246         int     lowest, highest, i;

1248         highest = apic_ipltopri[ipl] + APIC_VECTOR_MASK;
1249         lowest = apic_ipltopri[ipl - 1] + APIC_VECTOR_PER_IPL;

1251         if (highest < lowest) /* Both ipl and ipl - 1 map to same pri */
1252                 lowest -= APIC_VECTOR_PER_IPL;

1254 #ifdef  DEBUG
1255         if (apic_restrict_vector)       /* for testing shared interrupt logic */
1256                 highest = lowest + apic_restrict_vector + APIC_HI_PRI_VECTS;
1257 #endif /* DEBUG */
1258         if (pri == 0)
1259                 highest -= APIC_HI_PRI_VECTS;

1261         for (i = lowest; i <= highest; i++) {
1262                 if (APIC_CHECK_RESERVE_VECTORS(i))
1263                         continue;
1264                 if (apic_vector_to_irq[i] == APIC_RESV_IRQ) {
1265                         apic_vector_to_irq[i] = (uchar_t)irq;
1266                         ASSERT(i >= 0 && i <= UCHAR_MAX);
1267                         return ((uchar_t)i);
1262                         return (i);
1268                 }
1269         }

1271         return (0);
1272 }
```

*unchanged_portion_omitted_*

```
**********************************************************
   44887 Thu Sep  7 15:25:32 2017
new/usr/src/uts/i86pc/io/pcplusmp/apic_common.c
8626 make pcplusmp and apix warning-free
Reviewed by: Robert Mustacchi <rm@joyent.com>
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
**********************************************************
______unchanged_portion_omitted_

 845 int
 846 apic_cpu_add(psm_cpu_request_t *reqp)
 847 {
 848         int i, rv = 0;
 849         ulong_t iflag;
 850         boolean_t first = B_TRUE;
 851         uchar_t localver = 0;
 851         uchar_t localver;
 852         uint32_t localid, procid;
 853         processorid_t cpuid = (processorid_t)-1;
 854         mach_cpu_add_arg_t *ap;

 856         ASSERT(reqp != NULL);
 857         reqp->req.cpu_add.cpuid = (processorid_t)-1;

 859         /* Check whether CPU hotplug is supported. */
 860         if (!plat_dr_support_cpu() || apic_max_nproc == -1) {
 861                 return (ENOTSUP);
 862         }

 864         ap = (mach_cpu_add_arg_t *)reqp->req.cpu_add.argp;
 865         switch (ap->type) {
 866         case MACH_CPU_ARG_LOCAL_APIC:
 867                 localid = ap->arg.apic.apic_id;
 868                 procid = ap->arg.apic.proc_id;
 869                 if (localid >= 255 || procid > 255) {
 870                         cmn_err(CE_WARN,
 871                             "!apic: apicid(%u) or procid(%u) is invalid.",
 872                             localid, procid);
 873                         return (EINVAL);
 874                 }
 875                 break;

 877         case MACH_CPU_ARG_LOCAL_X2APIC:
 878                 localid = ap->arg.apic.apic_id;
 879                 procid = ap->arg.apic.proc_id;
 880                 if (localid >= UINT32_MAX) {
 881                         cmn_err(CE_WARN,
 882                             "!apic: x2apicid(%u) is invalid.", localid);
 883                         return (EINVAL);
 884                 } else if (localid >= 255 && apic_mode == LOCAL_APIC) {
 885                         cmn_err(CE_WARN, "!apic: system is in APIC mode, "
 886                             "can't support x2APIC processor.");
 887                         return (ENOTSUP);
 888                 }
 889                 break;

 891         default:
 892                 cmn_err(CE_WARN,
 893                     "!apic: unknown argument type %d to apic_cpu_add().",
 894                     ap->type);
 895                 return (EINVAL);
 896         }

 898         /* Use apic_ioapic_lock to sync with apic_get_next_bind_cpu. */
 899         iflag = intr_clear();
 900         lock_set(&apic_ioapic_lock);
```



```
 902         /* Check whether local APIC id already exists. */
 903         for (i = 0; i < apic_nproc; i++) {
 904                 if (!CPU_IN_SET(apic_cpumask, i))
 905                         continue;
 906                 if (apic_cpus[i].aci_local_id == localid) {
 907                         lock_clear(&apic_ioapic_lock);
 908                         intr_restore(iflag);
 909                         cmn_err(CE_WARN,
 910                             "!apic: local apic id %u already exists.",
 911                             localid);
 912                         return (EEXIST);
 913                 } else if (apic_cpus[i].aci_processor_id == procid) {
 914                         lock_clear(&apic_ioapic_lock);
 915                         intr_restore(iflag);
 916                         cmn_err(CE_WARN,
 917                             "!apic: processor id %u already exists.",
 918                             (int)procid);
 919                         return (EEXIST);
 920                 }

 922                 /*
 923                  * There's no local APIC version number available in MADT table,
 924                  * so assume that all CPUs are homogeneous and use local APIC
 925                  * version number of the first existing CPU.
 926                  */
 927                 if (first) {
 928                         first = B_FALSE;
 929                         localver = apic_cpus[i].aci_local_ver;
 930                 }
 931         }
 932         ASSERT(first == B_FALSE);

 934         /*
 935          * Try to assign the same cpuid if APIC id exists in the dirty cache.
 936          */
 937         for (i = 0; i < apic_max_nproc; i++) {
 938                 if (CPU_IN_SET(apic_cpumask, i)) {
 939                         ASSERT((apic_cpus[i].aci_status & APIC_CPU_FREE) == 0);
 940                         continue;
 941                 }
 942                 ASSERT(apic_cpus[i].aci_status & APIC_CPU_FREE);
 943                 if ((apic_cpus[i].aci_status & APIC_CPU_DIRTY) &&
 944                     apic_cpus[i].aci_local_id == localid &&
 945                     apic_cpus[i].aci_processor_id == procid) {
 946                         cpuid = i;
 947                         break;
 948                 }
 949         }

 951         /* Avoid the dirty cache and allocate fresh slot if possible. */
 952         if (cpuid == (processorid_t)-1) {
 953                 for (i = 0; i < apic_max_nproc; i++) {
 954                         if ((apic_cpus[i].aci_status & APIC_CPU_FREE) &&
 955                             (apic_cpus[i].aci_status & APIC_CPU_DIRTY) == 0) {
 956                                 cpuid = i;
 957                                 break;
 958                         }
 959                 }
 960         }

 962         /* Try to find any free slot as last resort. */
 963         if (cpuid == (processorid_t)-1) {
 964                 for (i = 0; i < apic_max_nproc; i++) {
 965                         if (apic_cpus[i].aci_status & APIC_CPU_FREE) {
 966                                 cpuid = i;
```

```
 967                                break;
 968                        }
 969                }
 970        }

 972        if (cpuid == (processorid_t)-1) {
 973                lock_clear(&apic_ioapic_lock);
 974                intr_restore(iflag);
 975                cmn_err(CE_NOTE,
 976                    "!apic: failed to allocate cpu id for processor %u.",
 977                    procid);
 978                rv = EAGAIN;
 979        } else if (ACPI_FAILURE(acpica_map_cpu(cpuid, procid))) {
 980                lock_clear(&apic_ioapic_lock);
 981                intr_restore(iflag);
 982                cmn_err(CE_NOTE,
 983                    "!apic: failed to build mapping for processor %u.",
 984                    procid);
 985                rv = EBUSY;
 986        } else {
 987                ASSERT(cpuid >= 0 && cpuid < NCPU);
 988                ASSERT(cpuid < apic_max_nproc && cpuid < max_ncpus);
 989                bzero(&apic_cpus[cpuid], sizeof (apic_cpus[0]));
 990                apic_cpus[cpuid].aci_processor_id = procid;
 991                apic_cpus[cpuid].aci_local_id = localid;
 992                apic_cpus[cpuid].aci_local_ver = localver;
 993                CPUSET_ATOMIC_ADD(apic_cpumask, cpuid);
 994                if (cpuid >= apic_nproc) {
 995                        apic_nproc = cpuid + 1;
 996                }
 997                lock_clear(&apic_ioapic_lock);
 998                intr_restore(iflag);
 999                reqp->req.cpu_add.cpuid = cpuid;
1000        }

1002        return (rv);
1003 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   26149 Thu Sep  7 15:25:32 2017
new/usr/src/uts/i86pc/io/pcplusmp/apic_introp.c
8626 make pcplusmp and apix warning-free
Reviewed by: Robert Mustacchi <rm@joyent.com>
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright (c) 2005, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright 2013 Pluribus Networks, Inc.
 * Copyright 2017 Joyent, Inc.
 */

/*
 * apic_introp.c:
 *      Has code for Advanced DDI interrupt framework support.
 */

#include <sys/cpuvar.h>
#include <sys/psm.h>
#include <sys/archsystm.h>
#include <sys/apic.h>
#include <sys/sunddi.h>
#include <sys/ddi_impldefs.h>
#include <sys/mach_intr.h>
#include <sys/sysmacros.h>
#include <sys/trap.h>
#include <sys/pci.h>
#include <sys/pci_intr_lib.h>
#include <sys/apic_common.h>

#define UCHAR_MAX       UINT8_MAX

extern struct av_head autovect[];

/*
 *      Local Function Prototypes
 */
apic_irq_t      *apic_find_irq(dev_info_t *, struct intrspec *, int);

/*
 * apic_pci_msi_enable_vector:
 *      Set the address/data fields in the MSI/X capability structure
 *      XXX: MSI-X support
 */
/* ARGSUSED */
```



```
void
apic_pci_msi_enable_vector(apic_irq_t *irq_ptr, int type, int inum, int vector,
    int count, int target_apic_id)
{
        uint64_t                msi_addr, msi_data;
        ushort_t                msi_ctrl;
        dev_info_t              *dip = irq_ptr->airq_dip;
        int                     cap_ptr = i_ddi_get_msi_msix_cap_ptr(dip);
        ddi_acc_handle_t        handle = i_ddi_get_pci_config_handle(dip);
        msi_regs_t              msi_regs;
        int                     irqno, i;
        void                    *intrmap_tbl[PCI_MSI_MAX_INTRS];

        DDI_INTR_IMPLDBG((CE_CONT, "apic_pci_msi_enable_vector: dip=0x%p\n"
            "\tdriver = %s, inum=0x%x vector=0x%x apicid=0x%x\n", (void *)dip,
            ddi_driver_name(dip), inum, vector, target_apic_id));

        ASSERT((handle != NULL) && (cap_ptr != 0));

        msi_regs.mr_data = vector;
        msi_regs.mr_addr = target_apic_id;

        for (i = 0; i < count; i++) {
                irqno = apic_vector_to_irq[vector + i];
                intrmap_tbl[i] = apic_irq_table[irqno]->airq_intrmap_private;
        }
        apic_vt_ops->apic_intrmap_alloc_entry(intrmap_tbl, dip, type,
            count, 0xff);
        for (i = 0; i < count; i++) {
                irqno = apic_vector_to_irq[vector + i];
                apic_irq_table[irqno]->airq_intrmap_private =
                    intrmap_tbl[i];
        }

        apic_vt_ops->apic_intrmap_map_entry(irq_ptr->airq_intrmap_private,
            (void *)&msi_regs, type, count);
        apic_vt_ops->apic_intrmap_record_msi(irq_ptr->airq_intrmap_private,
            &msi_regs);

        /* MSI Address */
        msi_addr = msi_regs.mr_addr;

        /* MSI Data: MSI is edge triggered according to spec */
        msi_data = msi_regs.mr_data;

        DDI_INTR_IMPLDBG((CE_CONT, "apic_pci_msi_enable_vector: addr=0x%lx "
            "data=0x%lx\n", (long)msi_addr, (long)msi_data));

        if (type == DDI_INTR_TYPE_MSI) {
                msi_ctrl = pci_config_get16(handle, cap_ptr + PCI_MSI_CTRL);

                /* Set the bits to inform how many MSIs are enabled */
                msi_ctrl |= ((highbit(count) -1) << PCI_MSI_MME_SHIFT);
                pci_config_put16(handle, cap_ptr + PCI_MSI_CTRL, msi_ctrl);

                /*
                 * Only set vector if not on hypervisor
                 */
                pci_config_put32(handle,
                    cap_ptr + PCI_MSI_ADDR_OFFSET, msi_addr);

                if (msi_ctrl &  PCI_MSI_64BIT_MASK) {
                        pci_config_put32(handle,
                            cap_ptr + PCI_MSI_ADDR_OFFSET + 4, msi_addr >> 32);
                        pci_config_put16(handle,
                            cap_ptr + PCI_MSI_64BIT_DATA, msi_data);
```

```
126                 } else {
127                         pci_config_put16(handle,
128                             cap_ptr + PCI_MSI_32BIT_DATA, msi_data);
129                 }

131         } else if (type == DDI_INTR_TYPE_MSIX) {
132                 uintptr_t       off;
133                 ddi_intr_msix_t *msix_p = i_ddi_get_msix(dip);

135                 ASSERT(msix_p != NULL);

137                 /* Offset into the "inum"th entry in the MSI-X table */
138                 off = (uintptr_t)msix_p->msix_tbl_addr +
139                     (inum  * PCI_MSIX_VECTOR_SIZE);

141                 ddi_put32(msix_p->msix_tbl_hdl,
142                     (uint32_t *)(off + PCI_MSIX_DATA_OFFSET), msi_data);
143                 ddi_put32(msix_p->msix_tbl_hdl,
144                     (uint32_t *)(off + PCI_MSIX_LOWER_ADDR_OFFSET), msi_addr);
145                 ddi_put32(msix_p->msix_tbl_hdl,
146                     (uint32_t *)(off + PCI_MSIX_UPPER_ADDR_OFFSET),
147                     msi_addr >> 32);
148         }
149 }
```

*unchanged_portion_omitted_*

```
188 /*
189  * Finds "count" contiguous MSI vectors starting at the proper alignment
190  * at "pri".
191  * Caller needs to make sure that count has to be power of 2 and should not
192  * be < 1.
193  */
194 uchar_t
195 apic_find_multi_vectors(int pri, int count)
196 {
197         int     lowest, highest, i, navail, start, msibits;

199         DDI_INTR_IMPLDBG((CE_CONT, "apic_find_mult: pri: %x, count: %x\n",
200             pri, count));

202         highest = apic_ipltopri[pri] + APIC_VECTOR_MASK;
203         lowest = apic_ipltopri[pri - 1] + APIC_VECTOR_PER_IPL;
204         navail = 0;

206         if (highest < lowest) /* Both ipl and ipl - 1 map to same pri */
207                 lowest -= APIC_VECTOR_PER_IPL;

209         /*
210          * msibits is the no. of lower order message data bits for the
211          * allocated MSI vectors and is used to calculate the aligned
212          * starting vector
213          */
214         msibits = count - 1;

216         /* It has to be contiguous */
217         for (i = lowest; i <= highest; i++) {
218                 navail = 0;

220                 /*
221                  * starting vector has to be aligned accordingly for
222                  * multiple MSIs
223                  */
224                 if (msibits)
225                         i = (i + msibits) & ~msibits;
226                 start = i;
227                 while ((apic_vector_to_irq[i] == APIC_RESV_IRQ) &&
```



```
228                     (i <= highest)) {
229                         if (APIC_CHECK_RESERVE_VECTORS(i))
230                                 break;
231                         navail++;
232                         if (navail >= count) {
233                                 ASSERT(start >= 0 && start <= UCHAR_MAX);
234                                 return ((uchar_t)start);
235                         }
229                         if (navail >= count)
230                                 return (start);
236                         i++;
237                 }
238         }
239         return (0);
240 }
```

*unchanged_portion_omitted_*

```
504 static int
505 apic_grp_set_cpu(int irqno, int new_cpu, int *result)
506 {
507         dev_info_t *orig_dip;
508         uint32_t orig_cpu;
509         ulong_t iflag;
510         apic_irq_t *irqps[PCI_MSI_MAX_INTRS];
511         int i;
512         int cap_ptr;
513         int msi_mask_off = 0;
508         int msi_mask_off;
514         ushort_t msi_ctrl;
515         uint32_t msi_pvm = 0;
510         uint32_t msi_pvm;
516         ddi_acc_handle_t handle;
517         int num_vectors = 0;
518         uint32_t vector;

520         DDI_INTR_IMPLDBG((CE_CONT, "APIC_GRP_SET_CPU\n"));

522         /*
523          * Take mutex to insure that table doesn't change out from underneath
524          * us while we're playing with it.
525          */
526         mutex_enter(&airq_mutex);
527         irqps[0] = apic_irq_table[irqno];
528         orig_cpu = irqps[0]->airq_temp_cpu;
529         orig_dip = irqps[0]->airq_dip;
530         num_vectors = irqps[0]->airq_intin_no;
531         vector = irqps[0]->airq_vector;

533         /* A "group" of 1 */
534         if (num_vectors == 1) {
535                 mutex_exit(&airq_mutex);
536                 return (apic_set_cpu(irqno, new_cpu, result));
537         }

539         *result = ENXIO;

541         if (irqps[0]->airq_mps_intr_index != MSI_INDEX) {
542                 mutex_exit(&airq_mutex);
543                 DDI_INTR_IMPLDBG((CE_CONT, "set_grp: intr not MSI\n"));
544                 goto set_grp_intr_done;
545         }
546         if ((num_vectors < 1) || ((num_vectors - 1) & vector)) {
547                 mutex_exit(&airq_mutex);
548                 DDI_INTR_IMPLDBG((CE_CONT,
549                     "set_grp: base vec not part of a grp or not aligned: "
550                     "vec:0x%x, num_vec:0x%x\n", vector, num_vectors));
```

```
551                 goto set_grp_intr_done;
552         }
553         DDI_INTR_IMPLDBG((CE_CONT, "set_grp: num intrs in grp: %d\n",
554             num_vectors));

556         ASSERT((num_vectors + vector) < APIC_MAX_VECTOR);

558         *result = EIO;

560         /*
561          * All IRQ entries in the table for the given device will be not
562          * shared.  Since they are not shared, the dip in the table will
563          * be true to the device of interest.
564          */
565         for (i = 1; i < num_vectors; i++) {
566                 irqps[i] = apic_irq_table[apic_vector_to_irq[vector + i]];
567                 if (irqps[i] == NULL) {
568                         mutex_exit(&airq_mutex);
569                         goto set_grp_intr_done;
570                 }
571 #ifdef DEBUG
572                 /* Sanity check: CPU and dip is the same for all entries. */
573                 if ((irqps[i]->airq_dip != orig_dip) ||
574                     (irqps[i]->airq_temp_cpu != orig_cpu)) {
575                         mutex_exit(&airq_mutex);
576                         DDI_INTR_IMPLDBG((CE_CONT,
577                             "set_grp: cpu or dip for vec 0x%x difft than for "
578                             "vec 0x%x\n", vector, vector + i));
579                         DDI_INTR_IMPLDBG((CE_CONT,
580                             "  cpu: %d vs %d, dip: 0x%p vs 0x%p\n", orig_cpu,
581                             irqps[i]->airq_temp_cpu, (void *)orig_dip,
582                             (void *)irqps[i]->airq_dip));
583                         goto set_grp_intr_done;
584                 }
585 #endif /* DEBUG */
586         }
587         mutex_exit(&airq_mutex);

589         cap_ptr = i_ddi_get_msi_msix_cap_ptr(orig_dip);
590         handle = i_ddi_get_pci_config_handle(orig_dip);
591         msi_ctrl = pci_config_get16(handle, cap_ptr + PCI_MSI_CTRL);

593         /* MSI Per vector masking is supported. */
594         if (msi_ctrl & PCI_MSI_PVM_MASK) {
595                 if (msi_ctrl &  PCI_MSI_64BIT_MASK)
596                         msi_mask_off = cap_ptr + PCI_MSI_64BIT_MASKBITS;
597                 else
598                         msi_mask_off = cap_ptr + PCI_MSI_32BIT_MASK;
599                 msi_pvm = pci_config_get32(handle, msi_mask_off);
600                 pci_config_put32(handle, msi_mask_off, (uint32_t)-1);
601                 DDI_INTR_IMPLDBG((CE_CONT,
602                     "set_grp: pvm supported.  Mask set to 0x%x\n",
603                     pci_config_get32(handle, msi_mask_off)));
604         }

606         iflag = intr_clear();
607         lock_set(&apic_ioapic_lock);

609         /*
610          * Do the first rebind and check for errors.  Apic_rebind_all returns
611          * an error if the CPU is not accepting interrupts.  If the first one
612          * succeeds they all will.
613          */
614         if (apic_rebind_all(irqps[0], new_cpu))
615                 (void) apic_rebind_all(irqps[0], orig_cpu);
616         else {
```



```
617                 irqps[0]->airq_cpu = new_cpu;

619                 for (i = 1; i < num_vectors; i++) {
620                         (void) apic_rebind_all(irqps[i], new_cpu);
621                         irqps[i]->airq_cpu = new_cpu;
622                 }
623                 *result = 0;    /* SUCCESS */
624         }

626         lock_clear(&apic_ioapic_lock);
627         intr_restore(iflag);

629         /* Reenable vectors if per vector masking is supported. */
630         if (msi_ctrl & PCI_MSI_PVM_MASK) {
631                 pci_config_put32(handle, msi_mask_off, msi_pvm);
632                 DDI_INTR_IMPLDBG((CE_CONT,
633                     "set_grp: pvm supported.  Mask restored to 0x%x\n",
634                     pci_config_get32(handle, msi_mask_off)));
635         }

637 set_grp_intr_done:
638         if (*result != 0)
639                 return (PSM_FAILURE);

641         return (PSM_SUCCESS);
642 }

644 int
645 apic_get_vector_intr_info(int vecirq, apic_get_intr_t *intr_params_p)
646 {
647         struct autovec *av_dev;
648         uchar_t irqno;
649         uint i;
644         int i;
650         apic_irq_t *irq_p;

652         /* Sanity check the vector/irq argument. */
653         ASSERT((vecirq >= 0) || (vecirq <= APIC_MAX_VECTOR));

655         mutex_enter(&airq_mutex);

657         /*
658          * Convert the vecirq arg to an irq using vector_to_irq table
659          * if the arg is a vector.  Pass thru if already an irq.
660          */
661         if ((intr_params_p->avgi_req_flags & PSMGI_INTRBY_FLAGS) ==
662             PSMGI_INTRBY_VEC)
663                 irqno = apic_vector_to_irq[vecirq];
664         else
665                 irqno = (uchar_t)vecirq;
660                 irqno = vecirq;

667         irq_p = apic_irq_table[irqno];

669         if ((irq_p == NULL) ||
670             ((irq_p->airq_mps_intr_index != RESERVE_INDEX) &&
671             ((irq_p->airq_temp_cpu == IRQ_UNBOUND) ||
672             (irq_p->airq_temp_cpu == IRQ_UNINIT)))) {
673                 mutex_exit(&airq_mutex);
674                 return (PSM_FAILURE);
675         }

677         if (intr_params_p->avgi_req_flags & PSMGI_REQ_CPUID) {

679                 /* Get the (temp) cpu from apic_irq table, indexed by irq. */
680                 intr_params_p->avgi_cpu_id = irq_p->airq_temp_cpu;
```

```
682                 /* Return user bound info for intrd. */
683                 if (intr_params_p->avgi_cpu_id & IRQ_USER_BOUND) {
684                         intr_params_p->avgi_cpu_id &= ~IRQ_USER_BOUND;
685                         intr_params_p->avgi_cpu_id |= PSMGI_CPU_USER_BOUND;
686                 }
687         }

689         if (intr_params_p->avgi_req_flags & PSMGI_REQ_VECTOR)
690                 intr_params_p->avgi_vector = irq_p->airq_vector;

692         if (intr_params_p->avgi_req_flags &
693             (PSMGI_REQ_NUM_DEVS | PSMGI_REQ_GET_DEVS))
694                 /* Get number of devices from apic_irq table shared field. */
695                 intr_params_p->avgi_num_devs = irq_p->airq_share;

697         if (intr_params_p->avgi_req_flags &  PSMGI_REQ_GET_DEVS) {

699                 intr_params_p->avgi_req_flags  |= PSMGI_REQ_NUM_DEVS;

701                 /* Some devices have NULL dip.  Don't count these. */
702                 if (intr_params_p->avgi_num_devs > 0) {
703                         for (i = 0, av_dev = autovect[irqno].avh_link;
704                             av_dev; av_dev = av_dev->av_link)
705                                 if (av_dev->av_vector && av_dev->av_dip)
706                                         i++;
707                         intr_params_p->avgi_num_devs =
708                             (uchar_t)MIN(intr_params_p->avgi_num_devs, i);
703                             MIN(intr_params_p->avgi_num_devs, i);
709                 }

711                 /* There are no viable dips to return. */
712                 if (intr_params_p->avgi_num_devs == 0)
713                         intr_params_p->avgi_dip_list = NULL;

715                 else {  /* Return list of dips */

717                         /* Allocate space in array for that number of devs. */
718                         intr_params_p->avgi_dip_list = kmem_zalloc(
719                             intr_params_p->avgi_num_devs *
720                             sizeof (dev_info_t *),
721                             KM_SLEEP);

723                         /*
724                          * Loop through the device list of the autovec table
725                          * filling in the dip array.
726                          *
727                          * Note that the autovect table may have some special
728                          * entries which contain NULL dips.  These will be
729                          * ignored.
730                          */
731                         for (i = 0, av_dev = autovect[irqno].avh_link;
732                             av_dev; av_dev = av_dev->av_link)
733                                 if (av_dev->av_vector && av_dev->av_dip)
734                                         intr_params_p->avgi_dip_list[i++] =
735                                             av_dev->av_dip;
736                 }
737         }

739         mutex_exit(&airq_mutex);

741         return (PSM_SUCCESS);
742 }
```

*_____unchanged_portion_omitted_*

```
**********************************************************
    2023 Thu Sep  7 15:25:33 2017
new/usr/src/uts/i86pc/pcplusmp/Makefile
8626 make pcplusmp and apix warning-free
Reviewed by: Robert Mustacchi <rm@joyent.com>
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
#
# uts/i86pc/pcplusmp/Makefile
#
# Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
# Copyright 2017, Joyent, Inc.
# Copyright 2016, Joyent, Inc.
#

#
#       This makefile drives the production of the pcplusmp "mach"
#       kernel module.
#
#       pcplusmp implementation architecture dependent
#

#
#       Path to the base of the uts directory tree (usually /usr/src/uts).
#
UTSBASE = ../..

#
#       Define the module and object file sets.
#
MODULE          = pcplusmp
OBJECTS         = $(PCPLUSMP_OBJS:%=$(OBJS_DIR)/%)
LINTS           = $(PCPLUSMP_OBJS:%.o=$(LINTS_DIR)/%.ln)
ROOTMODULE      = $(ROOT_PSM_MACH_DIR)/$(MODULE)

#
#       Include common rules.
#
include $(UTSBASE)/i86pc/Makefile.i86pc

#
#       Define targets
#
ALL_TARGET      = $(BINARY)
```



```
LINT_TARGET     = $(MODULE).lint
INSTALL_TARGET  = $(BINARY) $(ROOTMODULE)

DEBUG_FLGS      =
$(NOT_RELEASE_BUILD)DEBUG_DEFS  += $(DEBUG_FLGS)

#
# Depends on ACPI CA interpreter
#
LDFLAGS         += -dy -N misc/acpica

CERRWARN        += -_gcc=-Wno-unused-function

#
# For now, disable these lint checks; maintainers should endeavor
# to investigate and remove these for maximum lint coverage.
# Please do not carry these forward to new Makefiles.
#
LINTTAGS        += -erroff=E_BAD_PTR_CAST_ALIGN
LINTTAGS        += -erroff=E_SUPPRESSION_DIRECTIVE_UNUSED
LINTTAGS        += -erroff=E_STATIC_UNUSED
LINTTAGS        += -erroff=E_ASSIGN_NARROW_CONV

CERRWARN        += -_gcc=-Wno-uninitialized

#
#       Default build targets.
#
.KEEP_STATE:

def:            $(DEF_DEPS)

all:            $(ALL_DEPS)

clean:          $(CLEAN_DEPS)

clobber:        $(CLOBBER_DEPS)

lint:           $(LINT_DEPS)

modlintlib:     $(MODLINTLIB_DEPS)

clean.lint:     $(CLEAN_LINT_DEPS)

install:        $(INSTALL_DEPS)

#
#       Include common targets.
#
include $(UTSBASE)/i86pc/Makefile.targ
```

```
**********************************************************
   11666 Thu Sep  7 15:25:33 2017
new/usr/src/uts/i86pc/sys/apix.h
8626 make pcplusmp and apix warning-free
Reviewed by: Robert Mustacchi <rm@joyent.com>
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright (c) 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright 2017 Joyent, Inc.
 */

#ifndef __SYS_APIX_APIX_H
#define __SYS_APIX_APIX_H

#include <sys/note.h>
#include <sys/avintr.h>
#include <sys/traptrace.h>
#include <sys/apic.h>
#include <sys/apic_common.h>
#include <sys/apic_timer.h>

#ifdef  __cplusplus
extern  "C" {
#endif

#ifdef  DEBUG
#ifndef TRAPTRACE
#define TRAPTRACE
#endif
#endif

#define APIX_NAME               "apix"

#define APIX_NVECTOR            256     /* max number of per-cpu vectors */
#define APIX_NIRQ               256     /* maximum number of IRQs */
#define APIX_INVALID_VECT       0       /* invalid vector */

/* vector type */
#define APIX_TYPE_FIXED DDI_INTR_TYPE_FIXED     /* 1 */
#define APIX_TYPE_MSI           DDI_INTR_TYPE_MSI       /* 2 */
#define APIX_TYPE_MSIX  DDI_INTR_TYPE_MSIX      /* 4 */
#define APIX_TYPE_IPI           8

/* vector states */
enum {
```



```
        APIX_STATE_FREED = 0,
        APIX_STATE_OBSOLETED,   /* 1 */
        APIX_STATE_ALLOCED,     /* 2 */
        APIX_STATE_ENABLED,     /* 3 */
        APIX_STATE_DISABLED     /* 4 */
};
_____unchanged_portion_omitted_

extern struct apix_rebind_info apix_rebindinfo;

#define APIX_SET_REBIND_INFO(_ovp, _nvp)\
        if (((_ovp)->v_flags & APIX_VECT_MASKABLE) == 0) {\
                apix_rebindinfo.i_pri = (_ovp)->v_pri;\
                apix_rebindinfo.i_old_cpuid = (_ovp)->v_cpuid;\
                apix_rebindinfo.i_old_av = (_ovp)->v_autovect;\
                apix_rebindinfo.i_new_cpuid = (_nvp)->v_cpuid;\
                apix_rebindinfo.i_new_av = (_nvp)->v_autovect;\
                apix_rebindinfo.i_go = 1;\
        }

#define APIX_CLR_REBIND_INFO() \
        apix_rebindinfo.i_go = 0

#define APIX_IS_FAKE_INTR(_vector)\
        (apix_rebindinfo.i_go && (_vector) == APIX_RESV_VECTOR)

#define APIX_DO_FAKE_INTR(_cpu, _vector)\
        if (APIX_IS_FAKE_INTR(_vector)) {\
                struct autovec *tp = NULL;\
                struct autovec *tp;\
                if ((_cpu) == apix_rebindinfo.i_old_cpuid)\
                        tp = apix_rebindinfo.i_old_av;\
                else if ((_cpu) == apix_rebindinfo.i_new_cpuid)\
                        tp = apix_rebindinfo.i_new_av;\
                ASSERT(tp != NULL);\
                if (tp->av_vector != NULL &&\
                    (tp->av_flags & AV_PENTRY_PEND) == 0) {\
                        tp->av_flags |= AV_PENTRY_PEND;\
                        apix_insert_pending_av(apixs[(_cpu)], tp,\
                            tp->av_prilevel);\
                        apixs[(_cpu)]->x_intr_pending |=\
                            (1 << tp->av_prilevel);\
                }\
        }

extern int apix_add_avintr(void *intr_id, int ipl, avfunc xxintr, char *name,
    int vector, caddr_t arg1, caddr_t arg2, uint64_t *ticksp, dev_info_t *dip);
extern void apix_rem_avintr(void *intr_id, int ipl, avfunc xxintr,
    int virt_vect);

extern uint32_t apix_bind_cpu_locked(dev_info_t *dip);
extern apix_vector_t *apix_rebind(apix_vector_t *vecp, processorid_t tocpu,
    int count);

extern uchar_t apix_alloc_ipi(int ipl);
extern apix_vector_t *apix_alloc_intx(dev_info_t *dip, int inum, int irqno);
extern int apix_alloc_msi(dev_info_t *dip, int inum, int count, int behavior);
extern int apix_alloc_msix(dev_info_t *dip, int inum, int count, int behavior);
extern void apix_free_vectors(dev_info_t *dip, int inum, int count, int type);
extern void apix_enable_vector(apix_vector_t *vecp);
extern void apix_disable_vector(apix_vector_t *vecp);
extern int apix_obsolete_vector(apix_vector_t *vecp);
extern int apix_find_cont_vector_oncpu(uint32_t cpuid, int count);

extern void apix_set_dev_map(apix_vector_t *vecp, dev_info_t *dip, int inum);
extern apix_vector_t *apix_get_dev_map(dev_info_t *dip, int inum, int type);
```

```
328 extern apix_vector_t *apix_setup_io_intr(apix_vector_t *vecp);
329 extern void ioapix_init_intr(int mask_apic);
330 extern int apix_get_min_dev_inum(dev_info_t *dip, int type);
331 extern int apix_get_max_dev_inum(dev_info_t *dip, int type);

333 /*
334  * apix.c
335  */
336 extern int apix_addspl(int virtvec, int ipl, int min_ipl, int max_ipl);
337 extern int apix_delspl(int virtvec, int ipl, int min_ipl, int max_ipl);
338 extern void apix_intx_set_vector(int irqno, uint32_t cpuid, uchar_t vector);
339 extern apix_vector_t *apix_intx_get_vector(int irqno);
340 extern void apix_intx_enable(int irqno);
341 extern void apix_intx_disable(int irqno);
342 extern void apix_intx_free(int irqno);
343 extern int apix_intx_rebind(int irqno, processorid_t cpuid, uchar_t vector);
344 extern apix_vector_t *apix_set_cpu(apix_vector_t *vecp, int new_cpu,
345     int *result);
346 extern apix_vector_t *apix_grp_set_cpu(apix_vector_t *vecp, int new_cpu,
347     int *result);
348 extern void apix_level_intr_pre_eoi(int irq);
349 extern void apix_level_intr_post_dispatch(int irq);

351 #ifdef  __cplusplus
352 }
```

*unchanged_portion_omitted_*