

## [patch 8/8] 2.6.22-rc3 perfmon2 : IBS implementation for AMD64

---

*Source:* <http://linux.derkeiler.com/Mailing-Lists/Kernel/2007-06/msg06381.html>

---

- *From:* "Robert Richter" <[robert.richter@xxxxxxx](mailto:robert.richter@xxxxxxx)>
  - *Date:* Fri, 15 Jun 2007 19:02:11 +0200
- 

This patch renames module perfmon\_k8 to perfmon\_amd64.

Signed-off-by: Robert Richter <[robert.richter@xxxxxxx](mailto:robert.richter@xxxxxxx)>

Index: linux-2.6.22-rc3/arch/i386/perfmon/Kconfig

```
----- linux-2.6.22-rc3.orig/arch/i386/perfmon/Kconfig
+++ linux-2.6.22-rc3/arch/i386/perfmon/Kconfig
@@ -55,7 +55,7 @@ config I386_PERFMON_GEN_IA32
```

Enables 32-bit support for Intel architectural performance counters. Enable this option to support Intel Core Solo/Core Duo processors.

```
-config I386_PERFMON_K8
+config I386_PERFMON_AMD64
tristate "Support 32-bit mode AMD Athlon64/Opteron64 hardware performance counters"
depends on PERFMON
default n
```

Index: linux-2.6.22-rc3/arch/i386/perfmon/Makefile

```
----- linux-2.6.22-rc3.orig/arch/i386/perfmon/Makefile
+++ linux-2.6.22-rc3/arch/i386/perfmon/Makefile
@@ -8,7 +8,7 @@ obj-$(CONFIG_I386_PERFMON_P4) += perfmo
obj-$(CONFIG_I386_PERFMON_CORE) += perfmon_core.o
obj-$(CONFIG_I386_PERFMON_GEN_IA32) += perfmon_gen_ia32.o
obj-$(CONFIG_I386_PERFMON_PEBS) += perfmon_pebs_smpl.o
-obj-$(CONFIG_I386_PERFMON_K8) += perfmon_k8.o
+obj-$(CONFIG_I386_PERFMON_AMD64) += perfmon_amd64.o
```

```
-perfmon_k8-$(subst m,y,$(CONFIG_I386_PERFMON_K8)) += ../x86_64/perfmon/perfmon_k8.o
+perfmon_amd64-$(subst m,y,$(CONFIG_I386_PERFMON_AMD64)) +=
../x86_64/perfmon/perfmon_amd64.o
perfmon_core-$(subst m,y,$(CONFIG_I386_PERFMON_CORE)) += ../x86_64/perfmon/perfmon_core.o
```

Index: linux-2.6.22-rc3/arch/i386/perfmon/perfmon.c

```
----- linux-2.6.22-rc3.orig/arch/i386/perfmon/perfmon.c
+++ linux-2.6.22-rc3/arch/i386/perfmon/perfmon.c
@@ -931,7 +931,7 @@ fastcall void smp_pmu_interrupt(struct p
* 0 : no overflow
```

[patch 8/8] 2.6.22-rc3 perfmon2 : IBS implementation for AMD64

```
* 1 : at least one overflow
*
- * used by AMD K8 and Intel architectural PMU
+ * used by AMD64 and Intel architectural PMU
*/
static int __kprobes pfm_has_ovfl_p6(void)
{
@@ -1126,7 +1126,8 @@ int pfm_arch_pmu_config_init(struct _pfm
/*
* adjust stop routine based on PMU model
*
- * P6 : P6, Pentium M, AMD K8, CoreDuo/CoreSolo
+ * P6 : P6, Pentium M, CoreDuo/CoreSolo
+ * AMD64: AMD64 (K8, family 10h)
* P4 : Xeon, EM64T, P4
* CORE: Core 2,
*/
@@ -1279,7 +1280,7 @@ char *pfm_arch_get_pmu_module_name(void)
case 16:
/* All Opteron processors */
if (cpu_data->x86_vendor == X86_VENDOR_AMD)
- return "perfmon_k8";
+ return "perfmon_amd64";

switch(cpu_data->x86_model) {
case 0 ... 6:
Index: linux-2.6.22-rc3/arch/x86_64/perfmon/Kconfig
=====
--- linux-2.6.22-rc3.orig/arch/x86_64/perfmon/Kconfig
+++ linux-2.6.22-rc3/arch/x86_64/perfmon/Kconfig
@@ -14,7 +14,7 @@ config PERFMON_DEBUG
help
Enables perfmon debugging support

-config X86_64_PERFMON_K8
+config X86_64_PERFMON_AMD64
tristate "Support 64-bit mode AMD Athlon64 and Opteron64 hardware performance counters"
depends on PERFMON
default n
Index: linux-2.6.22-rc3/arch/x86_64/perfmon/Makefile
=====
--- linux-2.6.22-rc3.orig/arch/x86_64/perfmon/Makefile
+++ linux-2.6.22-rc3/arch/x86_64/perfmon/Makefile
@@ -4,7 +4,7 @@
#

obj-$(CONFIG_PERFMON) += perfmon.o
-obj-$(CONFIG_X86_64_PERFMON_K8) += perfmon_k8.o
+obj-$(CONFIG_X86_64_PERFMON_AMD64) += perfmon_amd64.o
obj-$(CONFIG_X86_64_PERFMON_P4) += perfmon_p4.o
obj-$(CONFIG_X86_64_PERFMON_CORE) += perfmon_core.o
```

obj-\$(CONFIG\_X86\_64\_PERFMON\_GEN\_IA32) += perfmon\_gen\_ia32.o  
Index: linux-2.6.22-rc3/arch/x86\_64/perfmon/perfmon\_amd64.c

```
=====
--- /dev/null
+++ linux-2.6.22-rc3/arch/x86_64/perfmon/perfmon_amd64.c
@@ -0,0 +1,483 @@
+/*
+ * This file contains the PMU description for the Ahtlon64 and Opteron64
+ * processors. It supports 32 and 64-bit modes.
+ *
+ * Copyright (c) 2005-2006 Hewlett-Packard Development Company, L.P.
+ * Contributed by Stephane Eranian <eranian@xxxxxxxxxxx>
+ *
+ * Copyright (c) 2007 Advanced Micro Devices, Inc.
+ * Contributed by Robert Richter <robert.richter@xxxxxxx>
+ *
+ * This program is free software; you can redistribute it and/or
+ * modify it under the terms of version 2 of the GNU General Public
+ * License as published by the Free Software Foundation.
+ *
+ * This program is distributed in the hope that it will be useful,
+ * but WITHOUT ANY WARRANTY; without even the implied warranty of
+ * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
+ * General Public License for more details.
+ *
+ * You should have received a copy of the GNU General Public License
+ * along with this program; if not, write to the Free Software
+ * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA
+ * 02111-1307 USA
+ */
+#include <linux/module.h>
+#include <linux/perfmon.h>
+#include <linux/vmalloc.h>
+#include <asm/nmi.h>
+
+MODULE_AUTHOR("Stephane Eranian <eranian@xxxxxxxxxxx>");
+MODULE_AUTHOR("Robert Richter <robert.richter@xxxxxxx>");
+MODULE_DESCRIPTION("AMD64 PMU description table");
+MODULE_LICENSE("GPL");
+
+static int force_nmi;
+MODULE_PARM_DESC(force_nmi, "bool: force use of NMI for PMU interrupt");
+module_param(force_nmi, bool, 0600);
+
+static struct pfm_arch_pmu_info pfm_amd64_pmu_info = {
+ .pmc_addrs = {
+ /* pmc0 */ { {MSR_K7_EVNTSEL0, 0}, 0, PFM_REGT_EN},
+ /* pmc1 */ { {MSR_K7_EVNTSEL1, 0}, 1, PFM_REGT_EN},
+ /* pmc2 */ { {MSR_K7_EVNTSEL2, 0}, 2, PFM_REGT_EN},
+ /* pmc3 */ { {MSR_K7_EVNTSEL3, 0}, 3, PFM_REGT_EN},
+ /* pmc4 */ { {MSR_AMD64_IBSFETCHCTL, 0}, 0, PFM_REGT_EN|PFM_REGT_IBS},

```

```

+/* pmc5 */ {{MSR_AMD64_IBSOPCTL, 0}, 0, PFM_REGT_EN|PFM_REGT_IBS},
+ },
+ .pmd_addrs = {
+/* pmd0 */ {{MSR_K7_PERFCTR0, 0}, 0, PFM_REGT_CTR},
+/* pmd1 */ {{MSR_K7_PERFCTR1, 0}, 0, PFM_REGT_CTR},
+/* pmd2 */ {{MSR_K7_PERFCTR2, 0}, 0, PFM_REGT_CTR},
+/* pmd3 */ {{MSR_K7_PERFCTR3, 0}, 0, PFM_REGT_CTR},
+/* pmd4 */ {{0, 0}, 0, PFM_REGT_CTR|PFM_REGT_IBS},
+/* pmd5 */ {{MSR_AMD64_IBSFETCHCTL, 0}, 0, PFM_REGT_IBS},
+/* pmd6 */ {{MSR_AMD64_IBSFETCHLINAD, 0}, 0, PFM_REGT_IBS},
+/* pmd7 */ {{MSR_AMD64_IBSFETCHPHYSAD, 0}, 0, PFM_REGT_IBS},
+/* pmd8 */ {{0, 0}, 0, PFM_REGT_CTR|PFM_REGT_IBS},
+/* pmd9 */ {{MSR_AMD64_IBSOPCTL, 0}, 0, PFM_REGT_IBS},
+/* pmd10 */ {{MSR_AMD64_IBSOPRIP, 0}, 0, PFM_REGT_IBS},
+/* pmd11 */ {{MSR_AMD64_IBSOPDATA, 0}, 0, PFM_REGT_IBS},
+/* pmd12 */ {{MSR_AMD64_IBSOPDATA2, 0}, 0, PFM_REGT_IBS},
+/* pmd13 */ {{MSR_AMD64_IBSOPDATA3, 0}, 0, PFM_REGT_IBS|PFM_REGT_IBS_EXT},
+/* pmd14 */ {{MSR_AMD64_IBSDCLINAD, 0}, 0, PFM_REGT_IBS|PFM_REGT_IBS_EXT},
+/* pmd15 */ {{MSR_AMD64_IBSDCPHYSAD, 0}, 0, PFM_REGT_IBS|PFM_REGT_IBS_EXT},
+ },
+ .pmu_style = PFM_X86_PMU_AMD64
+};
+
+/*
+ * force Local APIC interrupt on overflow
+ */
+#define PFM_K8_VAL (1ULL<<20)
+#define PFM_K8_NO64 (1ULL<<20)
+
+/*
+ * reserved bits must be zero
+ *
+ * - upper 32 bits are reserved
+ * - APIC enable bit is reserved (forced to 1)
+ * - bit 21 is reserved
+ */
+#define PFM_K8_RSVD (~(1ULL<<32)-1) \
+ | (1ULL<<20) \
+ | (1ULL<<21)
+
+/*
+ * We mark readonly bits as reserved and use the PMC for control
+ * operations only. Interrupt enable and clear bits are reserved too.
+ * IBSFETCHCTL is also implemented as PMD, where data can be read
+ * from. Same applies to IBSOPCTR.
+ */
+#define PFM_AMD64_IBSFETCHCTL_VAL PFM_AMD64_IBSFETCHEN
+#define PFM_AMD64_IBSFETCHCTL_NO64 PFM_AMD64_IBSFETCHEN
+#define PFM_AMD64_IBSFETCHCTL_RSVD (~(1ULL<<16)-1)
+#define PFM_AMD64_IBSOPCTL_VAL PFM_AMD64_IBSOPEN
+#define PFM_AMD64_IBSOPCTL_NO64 PFM_AMD64_IBSOPEN

```

[patch 8/8] 2.6.22-rc3 perfmon2 : IBS implementation for AMD64

```
+ #define PFM_AMD64_IBSOPCTL_RSVD (~((1ULL<<16)-1))
+
+ static struct pfm_reg_desc pfm_amd64_pmc_desc[]={
+ /* pmc0 */ PMC_D(PFM_REG_I64, "PERFSEL0", PFM_K8_VAL, PFM_K8_RSVD, PFM_K8_NO64,
+ MSR_K7_EVNTSEL0),
+ /* pmc1 */ PMC_D(PFM_REG_I64, "PERFSEL1", PFM_K8_VAL, PFM_K8_RSVD, PFM_K8_NO64,
+ MSR_K7_EVNTSEL1),
+ /* pmc2 */ PMC_D(PFM_REG_I64, "PERFSEL2", PFM_K8_VAL, PFM_K8_RSVD, PFM_K8_NO64,
+ MSR_K7_EVNTSEL2),
+ /* pmc3 */ PMC_D(PFM_REG_I64, "PERFSEL3", PFM_K8_VAL, PFM_K8_RSVD, PFM_K8_NO64,
+ MSR_K7_EVNTSEL3),
+ /* pmc4 */ PMC_D(PFM_REG_I, "IBSFETCHCTL", PFM_AMD64_IBSFETCHCTL_VAL,
+ PFM_AMD64_IBSFETCHCTL_RSVD, PFM_AMD64_IBSFETCHCTL_NO64,
+ MSR_AMD64_IBSFETCHCTL),
+ /* pmc5 */ PMC_D(PFM_REG_I, "IBSOPCTL", PFM_AMD64_IBSOPCTL_VAL,
+ PFM_AMD64_IBSOPCTL_RSVD, PFM_AMD64_IBSOPCTL_NO64, MSR_AMD64_IBSOPCTL),
+ };
+ #define PFM_AMD_NUM_PMCS ARRAY_SIZE(pfm_amd64_pmc_desc)
+
+ static struct pfm_reg_desc pfm_amd64_pmd_desc[] = {
+ /* pmd0 */ PMD_D(PFM_REG_C, "PERFCTR0", MSR_K7_PERFCTR0),
+ /* pmd1 */ PMD_D(PFM_REG_C, "PERFCTR1", MSR_K7_PERFCTR1),
+ /* pmd2 */ PMD_D(PFM_REG_C, "PERFCTR2", MSR_K7_PERFCTR2),
+ /* pmd3 */ PMD_D(PFM_REG_C, "PERFCTR3", MSR_K7_PERFCTR3),
+ /* pmd4 */ PMD_D(PFM_REG_ICV, "IBSFETCHCTR", PFM_VPMD_AMD64_IBSFETCHCTR),
+ /* pmd5 */ PMD_D(PFM_REG_IRO, "IBSFETCHCTL", MSR_AMD64_IBSFETCHCTL),
+ /* pmd6 */ PMD_D(PFM_REG_IRO, "IBSFETCHLINAD", MSR_AMD64_IBSFETCHLINAD),
+ /* pmd7 */ PMD_D(PFM_REG_IRO, "IBSFETCHPHYSAD", MSR_AMD64_IBSFETCHPHYSAD),
+ /* pmd8 */ PMD_D(PFM_REG_ICV, "IBSOPCTR", PFM_VPMD_AMD64_IBSOPCTR),
+ /* pmd9 */ PMD_D(PFM_REG_IRO, "IBSOPCTL", MSR_AMD64_IBSOPCTL),
+ /* pmd10 */ PMD_D(PFM_REG_IRO, "IBSOPRIP", MSR_AMD64_IBSOPRIP),
+ /* pmd11 */ PMD_D(PFM_REG_IRO, "IBSOPDATA", MSR_AMD64_IBSOPDATA),
+ /* pmd12 */ PMD_D(PFM_REG_IRO, "IBSOPDATA2", MSR_AMD64_IBSOPDATA2),
+ /* pmd13 */ PMD_D(PFM_REG_IRO, "IBSOPDATA3", MSR_AMD64_IBSOPDATA3),
+ /* pmd14 */ PMD_D(PFM_REG_IRO, "IBSDCLINAD", MSR_AMD64_IBSDCLINAD),
+ /* pmd15 */ PMD_D(PFM_REG_IRO, "IBSDCPHYSAD", MSR_AMD64_IBSDCPHYSAD),
+ };
+ #define PFM_AMD_NUM_PMDS ARRAY_SIZE(pfm_amd64_pmd_desc)
+
+ static struct pfm_context **pfm_nb_sys_owners;
+ static struct pfm_context *pfm_nb_task_owner;
+
+ static struct pfm_pmu_config pfm_amd64_pmu_conf;
+
+ /*
+ * There can only one user per socket for the Northbridge (NB) events
+ * so we enforce mutual exclusion as follows:
+ * - per-thread : only one context machine-wide can use NB events
+ * - system-wide: only one context per processor socket
+ *
+ * Exclusion is enforced at:
```

```

+ * - pfm_load_context()
+ * - pfm_write_pmcs() for attached contexts
+ *
+ * Exclusion is released at:
+ * - pfm_unload_context() or any calls that implicitly uses it
+ *
+ * return:
+ * 0 : successfully acquire NB access
+ * < 0: errno, failed to acquire NB access
+ */
+static int pfm_amd64_acquire_nb(struct pfm_context *ctx)
+{
+ struct pfm_context **entry, *old;
+ int proc_id;
+
+ #ifdef CONFIG_SMP
+ proc_id = topology_physical_package_id(smp_processor_id());
+ #else
+ proc_id = 0;
+ #endif
+
+ if (ctx->flags.system)
+ entry = &pfm_nb_sys_owners[proc_id];
+ else
+ entry = &pfm_nb_task_owner;
+
+ old = cmpxchg(entry, NULL, ctx);
+ if (!old) {
+ if (ctx->flags.system)
+ PFM_DBG("acquired Northbridge event access on socket %u", proc_id);
+ else
+ PFM_DBG("acquired Northbridge event access globally");
+ } else if (old != ctx) {
+ if (ctx->flags.system)
+ PFM_DBG("NorthBridge event conflict on socket %u", proc_id);
+ else
+ PFM_DBG("global NorthBridge event conflict");
+ return -EBUSY;
+ }
+ return 0;
+}
+
+/*
+ * invoked from pfm_write_pmcs() when pfm_nb_sys_owners is not NULL,i.e.,
+ * when we have detected a multi-core processor.
+ *
+ * context is locked, interrupts are masked
+ */
+static int pfm_amd64_pmc_write_check(struct pfm_context *ctx,
+ struct pfm_event_set *set,
+ struct pfarg_pmc *req)

```

```

+{
+ unsigned int event;
+ /*
+ * delay checking NB event until we load the context
+ */
+ if (ctx->state == PFM_CTX_UNLOADED)
+ return 0;
+
+ /*
+ * check event is NB event
+ */
+ event = (unsigned int)(req->reg_value & 0xff);
+ if (event < 0xee)
+ return 0;
+
+ return pfm_amd64_acquire_nb(ctx);
+}
+
+/*
+ * invoked on pfm_load_context().
+ * context is locked, interrupts are masked
+ */
+static int pfm_amd64_load_context(struct pfm_context *ctx)
+{
+ struct pfm_event_set *set;
+ unsigned int i, n;
+
+ /*
+ * scan all sets for NB events
+ */
+ list_for_each_entry(set, &ctx->list, list) {
+ n = set->nused_pmcs;
+ for(i=0; n; i++) {
+ if (!test_bit(i, ulp(set->used_pmcs)))
+ continue;
+ if ((set->pmcs[i] & 0xff) >= 0xee)
+ goto found;
+ n--;
+ }
+ }
+ return 0;
+found:
+ return pfm_amd64_acquire_nb(ctx);
+}
+
+/*
+ * invoked on pfm_unload_context()
+ */
+static int pfm_amd64_unload_context(struct pfm_context *ctx)
+{
+ struct pfm_context **entry, *old;

```

```

+ int proc_id;
+
+#ifdef CONFIG_SMP
+ proc_id = topology_physical_package_id(smp_processor_id());
+#else
+ proc_id = 0;
+#endif
+
+ /*
+ * unload always happens on the monitored CPU in system-wide
+ */
+ if (ctx->flags.system)
+ entry = &pfm_nb_sys_owners[proc_id];
+ else
+ entry = &pfm_nb_task_owner;
+
+ old = cmpxchg(entry, ctx, NULL);
+ if (old == ctx) {
+ if (ctx->flags.system)
+ PFM_DBG("released NorthBridge on socket %u", proc_id);
+ else
+ PFM_DBG("released NorthBridge events globally");
+ }
+ return 0;
+}
+
+ /*
+ * detect if we need to active NorthBridge event access control
+ */
+static int pfm_amd64_setup_nb_event_control(void)
+{
+ unsigned int c, n = 0;
+ unsigned int max_phys = 0;
+
+#ifdef CONFIG_SMP
+ for_each_present_cpu(c) {
+ if (cpu_data[c].phys_proc_id > max_phys)
+ max_phys = cpu_data[c].phys_proc_id;
+ }
+#else
+ max_phys = 0;
+#endif
+ if (max_phys > 255) {
+ PFM_INFO("socket id %d is too big to handle", max_phys);
+ return -ENOMEM;
+ }
+
+ n = max_phys + 1;
+ if (n < 2)
+ return 0;
+}

```

```

+ pfm_nb_sys_owners = vmalloc(n * sizeof(*pfm_nb_sys_owners));
+ if (!pfm_nb_sys_owners)
+ return -ENOMEM;
+
+ memset(pfm_nb_sys_owners, 0, n * sizeof(*pfm_nb_sys_owners));
+ pfm_nb_task_owner = NULL;
+
+ /*
+ * activate write-checker for PMC registers
+ */
+ for(c=0; c < PFM_AMD_NUM_PMCS; c++) {
+ pfm_amd64_pmc_desc[c].type |= PFM_REG_WC;
+ }
+
+ pfm_amd64_pmu_conf.load_context = pfm_amd64_load_context;
+ pfm_amd64_pmu_conf.unload_context = pfm_amd64_unload_context;
+ pfm_amd64_pmu_conf.pmc_write_check = pfm_amd64_pmc_write_check;
+
+ PFM_INFO("NorthBridge event access control enabled");
+
+ return 0;
+}
+
+static int pfm_amd64_detect_nmi(void)
+{
+ unsigned int i;
+
+ if (nmi_watchdog != NMI_LOCAL_APIC) {
+ if (force_nmi)
+ pfm_amd64_pmu_info.flags |= PFM_X86_FL_USE_NMI;
+ return 0;
+ }
+
+ /*
+ * assume NMI watchdog is initialized before PMU description module
+ * auto-detect which perfctr/eventsel is used by NMI watchdog
+ */
+ for (i=0; i < PFM_AMD_NUM_PMDS; i++) {
+ /* skip IBS registers */
+ if (pfm_amd64_pmu_info.pmc_addrs[i].reg_type & PFM_REGT_IBS)
+ continue;
+ if (avail_to_resrv_perfctr_nmi(pfm_amd64_pmd_desc[i].hw_addr))
+ continue;
+
+ PFM_INFO("NMI watchdog using %s/%s, disabling for perfmon",
+ pfm_amd64_pmc_desc[i].desc,
+ pfm_amd64_pmd_desc[i].desc);
+
+ pfm_amd64_pmc_desc[i].type = PFM_REG_NA;
+ pfm_amd64_pmd_desc[i].type = PFM_REG_NA;
+ pfm_amd64_pmu_info.pmc_addrs[i].reg_type = PFM_REGT_NA;

```

```

+ pfm_amd64_pmu_info.pmd_addrs[i].reg_type = PFM_REGT_NA;
+ }
+ pfm_amd64_pmu_info.flags |= PFM_X86_FL_USE_NMI;
+ return 0;
+}
+
+static int pfm_amd64_probe_pmu(void)
+{
+ if (current_cpu_data.x86_vendor != X86_VENDOR_AMD) {
+ PFM_INFO("not an AMD processor");
+ return -1;
+ }
+
+ switch (current_cpu_data.x86) {
+ case 15:
+ case 16:
+ PFM_INFO("found family=%d", current_cpu_data.x86);
+ break;
+ default:
+ PFM_INFO("unsupported family=%d", current_cpu_data.x86);
+ return -1;
+ }
+
+ /*
+ * check for local APIC (required)
+ */
+ if (!cpu_has_apic) {
+ PFM_INFO("no local APIC, unsupported");
+ return -1;
+ }
+ if (pfm_amd64_detect_nmi()) {
+ PFM_INFO("NMI detection failed");
+ return -1;
+ }
+ if (current_cpu_data.x86_max_cores > 1)
+ pfm_amd64_setup_nb_event_control();
+
+ PFM_INFO("Using AMD64 PMU");
+ if (pfm_amd64_pmu_info.flags & PFM_X86_FL_IBS)
+ PFM_INFO("IBS is supported by processor");
+ if (pfm_amd64_pmu_info.flags & PFM_X86_FL_IBS_EXT)
+ PFM_INFO("IBS extended registers are supported by processor");
+
+ return 0;
+}
+
+static inline void
+pfm_amd64_check_register(struct pfm_pmu_config *cfg,
+ struct pfm_reg_desc *reg,
+ struct pfm_arch_ext_reg *ext_reg)
+{

```

```

+ struct pfm_arch_pmu_info *arch_info = cfg->arch_info;
+
+ if (!ext_reg->reg_type & PFM_REGT_AMD64)
+ /* No special AMD64 PMU register */
+ return;
+
+ /* Disable register */
+ reg->type &= ~PFM_REG_I;
+
+ switch (ext_reg->reg_type & PFM_REGT_AMD64) {
+ case (PFM_REGT_IBS):
+ /* IBS register */
+ if (!(arch_info->flags & PFM_X86_FL_IBS))
+ return;
+ break;
+ case (PFM_REGT_IBS|PFM_REGT_IBS_EXT):
+ /* IBS extended register */
+ if (!(arch_info->flags & PFM_X86_FL_IBS_EXT))
+ return;
+ break;
+ default:
+ return;
+ }
+
+ /* Enable register */
+ reg->type |= PFM_REG_I;
+}
+
+static void pfm_amd64_setup_pmu(struct pfm_pmu_config *cfg)
+{
+ u16 i;
+ struct pfm_arch_pmu_info *arch_info = cfg->arch_info;
+
+ /* set PMU features depending on CPUID */
+ arch_info->flags &= ~(PFM_X86_FL_IBS|PFM_X86_FL_IBS_EXT);
+ switch (current_cpu_data.x86) {
+ case 15:
+ break;
+ case 16:
+ arch_info->flags |= PFM_X86_FL_IBS;
+ break;
+ default:
+ break;
+ }
+
+ /* Disable unsupported PMC/PMD registers */
+ for (i = 0; i < cfg->num_pmc_entries; i++) {
+ pfm_amd64_check_register(cfg, &cfg->pmc_desc[i],
+ &arch_info->pmc_addrs[i]);
+ }
+ for (i = 0; i < cfg->num_pmd_entries; i++) {

```

```

+ pfm_amd64_check_register(cfg, &cfg->pmd_desc[i],
+ &arch_info->pmd_addrs[i]);
+ }
+ }
+
+static struct pfm_pmu_config pfm_amd64_pmu_conf = {
+ .pmu_name = "AMD64",
+ .counter_width = 47,
+ .pmd_desc = pfm_amd64_pmd_desc,
+ .pmc_desc = pfm_amd64_pmc_desc,
+ .num_pmc_entries = PFM_AMD_NUM_PMCS,
+ .num_pmd_entries = PFM_AMD_NUM_PMDS,
+ .probe_pmu = pfm_amd64_probe_pmu,
+ .version = "1.2",
+ .arch_info = &pfm_amd64_pmu_info,
+ .flags = PFM_PMU_BUILTIN_FLAG,
+ .owner = THIS_MODULE,
+ .pmd_sread = pfm_pmd_sread,
+ .pmd_swrite = pfm_pmd_swrite,
+ };
+
+static int __init pfm_amd64_pmu_init_module(void)
+{
+ pfm_amd64_setup_pmu(&pfm_amd64_pmu_conf);
+ return pfm_pmu_register(&pfm_amd64_pmu_conf);
+ }
+
+static void __exit pfm_amd64_pmu_cleanup_module(void)
+{
+ if (pfm_nb_sys_owners)
+ vfree(pfm_nb_sys_owners);
+
+ pfm_pmu_unregister(&pfm_amd64_pmu_conf);
+ }
+
+module_init(pfm_amd64_pmu_init_module);
+module_exit(pfm_amd64_pmu_cleanup_module);
Index: linux-2.6.22-rc3/arch/x86_64/perfmon/perfmon_k8.c

```

```

=====
--- linux-2.6.22-rc3.orig/arch/x86_64/perfmon/perfmon_k8.c
+++ /dev/null
@@ -1,483 +0,0 @@
-/*
- * This file contains the PMU description for the Ahtlon64 and Opteron64
- * processors. It supports 32 and 64-bit modes.
- *
- * Copyright (c) 2005-2006 Hewlett-Packard Development Company, L.P.
- * Contributed by Stephane Eranian <eranian@xxxxxxxxxxx>
- *
- * Copyright (c) 2007 Advanced Micro Devices, Inc.
- * Contributed by Robert Richter <robert.richter@xxxxxxxx>

```

```

- *
- * This program is free software; you can redistribute it and/or
- * modify it under the terms of version 2 of the GNU General Public
- * License as published by the Free Software Foundation.
- *
- * This program is distributed in the hope that it will be useful,
- * but WITHOUT ANY WARRANTY; without even the implied warranty of
- * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
- * General Public License for more details.
- *
- * You should have received a copy of the GNU General Public License
- * along with this program; if not, write to the Free Software
- * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA
- * 02111-1307 USA
- */
#include <linux/module.h>
#include <linux/perfmon.h>
#include <linux/vmalloc.h>
#include <asm/nmi.h>
-
MODULE_AUTHOR("Stephane Eranian <eranian@xxxxxxxxxx>");
MODULE_AUTHOR("Robert Richter <robert.richter@xxxxxxx>");
MODULE_DESCRIPTION("AMD64 PMU description table");
MODULE_LICENSE("GPL");
-
static int force_nmi;
MODULE_PARM_DESC(force_nmi, "bool: force use of NMI for PMU interrupt");
module_param(force_nmi, bool, 0600);
-
static struct pfm_arch_pmu_info pfm_amd64_pmu_info = {
- .pmc_addrs = {
-/* pmc0 */ {{MSR_K7_EVNTSEL0, 0}, 0, PFM_REGT_EN},
-/* pmc1 */ {{MSR_K7_EVNTSEL1, 0}, 1, PFM_REGT_EN},
-/* pmc2 */ {{MSR_K7_EVNTSEL2, 0}, 2, PFM_REGT_EN},
-/* pmc3 */ {{MSR_K7_EVNTSEL3, 0}, 3, PFM_REGT_EN},
-/* pmc4 */ {{MSR_AMD64_IBSFETCHCTL, 0}, 0, PFM_REGT_EN|PFM_REGT_IBS},
-/* pmc5 */ {{MSR_AMD64_IBSOPCTL, 0}, 0, PFM_REGT_EN|PFM_REGT_IBS},
- },
- .pmd_addrs = {
-/* pmd0 */ {{MSR_K7_PERFCTR0, 0}, 0, PFM_REGT_CTR},
-/* pmd1 */ {{MSR_K7_PERFCTR1, 0}, 0, PFM_REGT_CTR},
-/* pmd2 */ {{MSR_K7_PERFCTR2, 0}, 0, PFM_REGT_CTR},
-/* pmd3 */ {{MSR_K7_PERFCTR3, 0}, 0, PFM_REGT_CTR},
-/* pmd4 */ {{0, 0}, 0, PFM_REGT_CTR|PFM_REGT_IBS},
-/* pmd5 */ {{MSR_AMD64_IBSFETCHCTL, 0}, 0, PFM_REGT_IBS},
-/* pmd6 */ {{MSR_AMD64_IBSFETCHLINAD, 0}, 0, PFM_REGT_IBS},
-/* pmd7 */ {{MSR_AMD64_IBSFETCHPHYSAD, 0}, 0, PFM_REGT_IBS},
-/* pmd8 */ {{0, 0}, 0, PFM_REGT_CTR|PFM_REGT_IBS},
-/* pmd9 */ {{MSR_AMD64_IBSOPCTL, 0}, 0, PFM_REGT_IBS},
-/* pmd10 */ {{MSR_AMD64_IBSOPRIP, 0}, 0, PFM_REGT_IBS},
-/* pmd11 */ {{MSR_AMD64_IBSOPDATA, 0}, 0, PFM_REGT_IBS},

```

[patch 8/8] 2.6.22-rc3 perfmon2 : IBS implementation for AMD64

```

-/* pmd12 */ {{MSR_AMD64_IBSOPDATA2, 0}, 0, PFM_REGT_IBS},
-/* pmd13 */ {{MSR_AMD64_IBSOPDATA3, 0}, 0, PFM_REGT_IBS|PFM_REGT_IBS_EXT},
-/* pmd14 */ {{MSR_AMD64_IBSDCLINAD, 0}, 0, PFM_REGT_IBS|PFM_REGT_IBS_EXT},
-/* pmd15 */ {{MSR_AMD64_IBSDCPHYSAD, 0}, 0, PFM_REGT_IBS|PFM_REGT_IBS_EXT},
- },
- .pmu_style = PFM_X86_PMU_AMD64
-};
-
-/*
- * force Local APIC interrupt on overflow
- */
-#define PFM_K8_VAL (1ULL<<20)
-#define PFM_K8_NO64 (1ULL<<20)
-
-/*
- * reserved bits must be zero
- *
- * - upper 32 bits are reserved
- * - APIC enable bit is reserved (forced to 1)
- * - bit 21 is reserved
- */
-#define PFM_K8_RSVD (~(1ULL<<32)-1) \
- | (1ULL<<20) \
- | (1ULL<<21)
-
-/*
- * We mark readonly bits as reserved and use the PMC for control
- * operations only. Interrupt enable and clear bits are reserved too.
- * IBSFETCHCTL is also implemented as PMD, where data can be read
- * from. Same applies to IBSOPCTR.
- */
-#define PFM_AMD64_IBSFETCHCTL_VAL PFM_AMD64_IBSFETCHEN
-#define PFM_AMD64_IBSFETCHCTL_NO64 PFM_AMD64_IBSFETCHEN
-#define PFM_AMD64_IBSFETCHCTL_RSVD (~(1ULL<<16)-1)
-#define PFM_AMD64_IBSOPCTL_VAL PFM_AMD64_IBSOPEN
-#define PFM_AMD64_IBSOPCTL_NO64 PFM_AMD64_IBSOPEN
-#define PFM_AMD64_IBSOPCTL_RSVD (~(1ULL<<16)-1)
-
-static struct pfm_reg_desc pfm_amd64_pmc_desc[]={
-/* pmc0 */ PMC_D(PFM_REG_I64, "PERFSEL0", PFM_K8_VAL, PFM_K8_RSVD, PFM_K8_NO64,
MSR_K7_EVNTSEL0),
-/* pmc1 */ PMC_D(PFM_REG_I64, "PERFSEL1", PFM_K8_VAL, PFM_K8_RSVD, PFM_K8_NO64,
MSR_K7_EVNTSEL1),
-/* pmc2 */ PMC_D(PFM_REG_I64, "PERFSEL2", PFM_K8_VAL, PFM_K8_RSVD, PFM_K8_NO64,
MSR_K7_EVNTSEL2),
-/* pmc3 */ PMC_D(PFM_REG_I64, "PERFSEL3", PFM_K8_VAL, PFM_K8_RSVD, PFM_K8_NO64,
MSR_K7_EVNTSEL3),
-/* pmc4 */ PMC_D(PFM_REG_I, "IBSFETCHCTL", PFM_AMD64_IBSFETCHCTL_VAL,
PFM_AMD64_IBSFETCHCTL_RSVD, PFM_AMD64_IBSFETCHCTL_NO64,
MSR_AMD64_IBSFETCHCTL),
-/* pmc5 */ PMC_D(PFM_REG_I, "IBSOPCTL", PFM_AMD64_IBSOPCTL_VAL,
```

```

PFM_AMD64_IBSOPCTL_RSVD, PFM_AMD64_IBSOPCTL_NO64, MSR_AMD64_IBSOPCTL),
-};
-#define PFM_AMD_NUM_PMCS ARRAY_SIZE(pfm_amd64_pmc_desc)
-
-static struct pfm_reg_desc pfm_amd64_pmd_desc[] = {
-/* pmd0 */ PMD_D(PFM_REG_C, "PERFCTR0", MSR_K7_PERFCTR0),
-/* pmd1 */ PMD_D(PFM_REG_C, "PERFCTR1", MSR_K7_PERFCTR1),
-/* pmd2 */ PMD_D(PFM_REG_C, "PERFCTR2", MSR_K7_PERFCTR2),
-/* pmd3 */ PMD_D(PFM_REG_C, "PERFCTR3", MSR_K7_PERFCTR3),
-/* pmd4 */ PMD_D(PFM_REG_ICV, "IBSFETCHCTR", PFM_VPMD_AMD64_IBSFETCHCTR),
-/* pmd5 */ PMD_D(PFM_REG_IRO, "IBSFETCHCTL", MSR_AMD64_IBSFETCHCTL),
-/* pmd6 */ PMD_D(PFM_REG_IRO, "IBSFETCHLINAD", MSR_AMD64_IBSFETCHLINAD),
-/* pmd7 */ PMD_D(PFM_REG_IRO, "IBSFETCHPHYSAD", MSR_AMD64_IBSFETCHPHYSAD),
-/* pmd8 */ PMD_D(PFM_REG_ICV, "IBSOPCTR", PFM_VPMD_AMD64_IBSOPCTR),
-/* pmd9 */ PMD_D(PFM_REG_IRO, "IBSOPCTL", MSR_AMD64_IBSOPCTL),
-/* pmd10 */ PMD_D(PFM_REG_IRO, "IBSOPRIP", MSR_AMD64_IBSOPRIP),
-/* pmd11 */ PMD_D(PFM_REG_IRO, "IBSOPDATA", MSR_AMD64_IBSOPDATA),
-/* pmd12 */ PMD_D(PFM_REG_IRO, "IBSOPDATA2", MSR_AMD64_IBSOPDATA2),
-/* pmd13 */ PMD_D(PFM_REG_IRO, "IBSOPDATA3", MSR_AMD64_IBSOPDATA3),
-/* pmd14 */ PMD_D(PFM_REG_IRO, "IBSDCLINAD", MSR_AMD64_IBSDCLINAD),
-/* pmd15 */ PMD_D(PFM_REG_IRO, "IBSDCPHYSAD", MSR_AMD64_IBSDCPHYSAD),
-};
-#define PFM_AMD_NUM_PMDS ARRAY_SIZE(pfm_amd64_pmd_desc)
-
-static struct pfm_context **pfm_nb_sys_owners;
-static struct pfm_context *pfm_nb_task_owner;
-
-static struct pfm_pmu_config pfm_amd64_pmu_conf;
-
-/*
- * There can only one user per socket for the Northbridge (NB) events
- * so we enforce mutual exclusion as follows:
- * - per-thread : only one context machine-wide can use NB events
- * - system-wide: only one context per processor socket
- *
- * Exclusion is enforced at:
- * - pfm_load_context()
- * - pfm_write_pmcs() for attached contexts
- *
- * Exclusion is released at:
- * - pfm_unload_context() or any calls that implicetly uses it
- *
- * return:
- * 0 : successfully acquire NB access
- * < 0: errno, failed to acquire NB access
- */
-static int pfm_amd64_acquire_nb(struct pfm_context *ctx)
-{
- struct pfm_context **entry, *old;
- int proc_id;
-

```

```

-#ifdef CONFIG_SMP
- proc_id = topology_physical_package_id(smp_processor_id());
-#else
- proc_id = 0;
-#endif
-
- if (ctx->flags.system)
- entry = &pfm_nb_sys_owners[proc_id];
- else
- entry = &pfm_nb_task_owner;
-
- old = cmpxchg(entry, NULL, ctx);
- if (!old) {
- if (ctx->flags.system)
- PFM_DBG("acquired Northbridge event access on socket %u", proc_id);
- else
- PFM_DBG("acquired Northbridge event access globally");
- } else if (old != ctx) {
- if (ctx->flags.system)
- PFM_DBG("NorthBridge event conflict on socket %u", proc_id);
- else
- PFM_DBG("global NorthBridge event conflict");
- return -EBUSY;
- }
- return 0;
-}
-
-/*
- * invoked from pfm_write_pmcs() when pfm_nb_sys_owners is not NULL,i.e.,
- * when we have detected a multi-core processor.
- *
- * context is locked, interrupts are masked
- */
-static int pfm_amd64_pmc_write_check(struct pfm_context *ctx,
- struct pfm_event_set *set,
- struct pfarg_pmc *req)
-{
- unsigned int event;
- /*
- * delay checking NB event until we load the context
- */
- if (ctx->state == PFM_CTX_UNLOADED)
- return 0;
-
- /*
- * check event is NB event
- */
- event = (unsigned int)(req->reg_value & 0xff);
- if (event < 0xee)
- return 0;
-
-

```

```

- return pfm_amd64_acquire_nb(ctx);
- }
-
- /*
-  * invoked on pfm_load_context().
-  * context is locked, interrupts are masked
-  */
-static int pfm_amd64_load_context(struct pfm_context *ctx)
- {
-     struct pfm_event_set *set;
-     unsigned int i, n;
-
-     /*
-      * scan all sets for NB events
-      */
-     list_for_each_entry(set, &ctx->list, list) {
-         n = set->nused_pmcs;
-         for(i=0; n; i++) {
-             if (!test_bit(i, ulp(set->used_pmcs)))
-                 continue;
-             if ((set->pmcs[i] & 0xff) >= 0xee)
-                 goto found;
-             n--;
-         }
-     }
-     return 0;
- found:
-     return pfm_amd64_acquire_nb(ctx);
- }
-
- /*
-  * invoked on pfm_unload_context()
-  */
-static int pfm_amd64_unload_context(struct pfm_context *ctx)
- {
-     struct pfm_context **entry, *old;
-     int proc_id;
-
-     #ifdef CONFIG_SMP
-     proc_id = topology_physical_package_id(smp_processor_id());
-     #else
-     proc_id = 0;
-     #endif
-
-     /*
-      * unload always happens on the monitored CPU in system-wide
-      */
-     if (ctx->flags.system)
-         entry = &pfm_nb_sys_owners[proc_id];
-     else
-         entry = &pfm_nb_task_owner;

```

```

-
- old = cmpxchg(entry, ctx, NULL);
- if (old == ctx) {
- if (ctx->flags.system)
- PFM_DBG("released NorthBridge on socket %u", proc_id);
- else
- PFM_DBG("released NorthBridge events globally");
- }
- return 0;
-}
-
-/*
- * detect if we need to active NorthBridge event access control
- */
-static int pfm_amd64_setup_nb_event_control(void)
-{
- unsigned int c, n = 0;
- unsigned int max_phys = 0;
-
-#ifdef CONFIG_SMP
- for_each_present_cpu(c) {
- if (cpu_data[c].phys_proc_id > max_phys)
- max_phys = cpu_data[c].phys_proc_id;
- }
-#else
- max_phys = 0;
-#endif
- if (max_phys > 255) {
- PFM_INFO("socket id %d is too big to handle", max_phys);
- return -ENOMEM;
- }
-
- n = max_phys + 1;
- if (n < 2)
- return 0;
-
- pfm_nb_sys_owners = vmalloc(n * sizeof(*pfm_nb_sys_owners));
- if (!pfm_nb_sys_owners)
- return -ENOMEM;
-
- memset(pfm_nb_sys_owners, 0, n * sizeof(*pfm_nb_sys_owners));
- pfm_nb_task_owner = NULL;
-
- /*
- * activate write-checker for PMC registers
- */
- for(c=0; c < PFM_AMD_NUM_PMCS; c++) {
- pfm_amd64_pmc_desc[c].type |= PFM_REG_WC;
- }
-
- pfm_amd64_pmu_conf.load_context = pfm_amd64_load_context;

```

```

- pfm_amd64_pmu_conf.unload_context = pfm_amd64_unload_context;
- pfm_amd64_pmu_conf.pmc_write_check = pfm_amd64_pmc_write_check;
-
- PFM_INFO("NorthBridge event access control enabled");
-
- return 0;
-}
-
-static int pfm_amd64_detect_nmi(void)
-{
- unsigned int i;
-
- if (nmi_watchdog != NMI_LOCAL_APIC) {
- if (force_nmi)
- pfm_amd64_pmu_info.flags |= PFM_X86_FL_USE_NMI;
- return 0;
- }
-
- /*
- * assume NMI watchdog is initialized before PMU description module
- * auto-detect which perfctr/eventsel is used by NMI watchdog
- */
- for (i=0; i < PFM_AMD_NUM_PMDS; i++) {
- /* skip IBS registers */
- if (pfm_amd64_pmu_info.pmc_addrs[i].reg_type & PFM_REGT_IBS)
- continue;
- if (avail_to_resrv_perfctr_nmi(pfm_amd64_pmd_desc[i].hw_addr))
- continue;
-
- PFM_INFO("NMI watchdog using %s/%s, disabling for perfmon",
- pfm_amd64_pmc_desc[i].desc,
- pfm_amd64_pmd_desc[i].desc);
-
- pfm_amd64_pmc_desc[i].type = PFM_REG_NA;
- pfm_amd64_pmd_desc[i].type = PFM_REG_NA;
- pfm_amd64_pmu_info.pmc_addrs[i].reg_type = PFM_REGT_NA;
- pfm_amd64_pmu_info.pmd_addrs[i].reg_type = PFM_REGT_NA;
- }
- pfm_amd64_pmu_info.flags |= PFM_X86_FL_USE_NMI;
- return 0;
-}
-
-static int pfm_amd64_probe_pmu(void)
-{
- if (current_cpu_data.x86_vendor != X86_VENDOR_AMD) {
- PFM_INFO("not an AMD processor");
- return -1;
- }
-
- switch (current_cpu_data.x86) {
- case 15:

```

```

- case 16:
- PFM_INFO("found family=%d", current_cpu_data.x86);
- break;
- default:
- PFM_INFO("unsupported family=%d", current_cpu_data.x86);
- return -1;
- }
-
- /*
- * check for local APIC (required)
- */
- if (!cpu_has_apic) {
- PFM_INFO("no local APIC, unsupported");
- return -1;
- }
- if (pfm_amd64_detect_nmi()) {
- PFM_INFO("NMI detection failed");
- return -1;
- }
- if (current_cpu_data.x86_max_cores > 1)
- pfm_amd64_setup_nb_event_control();
-
- PFM_INFO("Using AMD64 PMU");
- if (pfm_amd64_pmu_info.flags & PFM_X86_FL_IBS)
- PFM_INFO("IBS is supported by processor");
- if (pfm_amd64_pmu_info.flags & PFM_X86_FL_IBS_EXT)
- PFM_INFO("IBS extended registers are supported by processor");
-
- return 0;
-}
-
-static inline void
-pfm_amd64_check_register(struct pfm_pmu_config *cfg,
- struct pfm_reg_desc *reg,
- struct pfm_arch_ext_reg *ext_reg)
-{
- struct pfm_arch_pmu_info *arch_info = cfg->arch_info;
-
- if (!(ext_reg->reg_type & PFM_REGT_AMD64))
- /* No special AMD64 PMU register */
- return;
-
- /* Disable register */
- reg->type &= ~PFM_REG_I;
-
- switch (ext_reg->reg_type & PFM_REGT_AMD64) {
- case (PFM_REGT_IBS):
- /* IBS register */
- if (!(arch_info->flags & PFM_X86_FL_IBS))
- return;
- break;

```

```

- case (PFM_REGT_IBS|PFM_REGT_IBS_EXT):
- /* IBS extended register */
- if (!(arch_info->flags & PFM_X86_FL_IBS_EXT))
- return;
- break;
- default:
- return;
- }
-
- /* Enable register */
- reg->type |= PFM_REG_I;
- }
-
-static void pfm_amd64_setup_pmu(struct pfm_pmu_config *cfg)
-{
- u16 i;
- struct pfm_arch_pmu_info *arch_info = cfg->arch_info;
-
- /* set PMU features depending on CPUID */
- arch_info->flags &= ~(PFM_X86_FL_IBS|PFM_X86_FL_IBS_EXT);
- switch (current_cpu_data.x86) {
- case 15:
- break;
- case 16:
- arch_info->flags |= PFM_X86_FL_IBS;
- break;
- default:
- break;
- }
-
- /* Disable unsupported PMC/PMD registers */
- for (i = 0; i < cfg->num_pmc_entries; i++) {
- pfm_amd64_check_register(cfg, &cfg->pmc_desc[i],
- &arch_info->pmc_addrs[i]);
- }
- for (i = 0; i < cfg->num_pmd_entries; i++) {
- pfm_amd64_check_register(cfg, &cfg->pmd_desc[i],
- &arch_info->pmd_addrs[i]);
- }
- }
-
-static struct pfm_pmu_config pfm_amd64_pmu_conf = {
- .pmu_name = "AMD64",
- .counter_width = 47,
- .pmd_desc = pfm_amd64_pmd_desc,
- .pmc_desc = pfm_amd64_pmc_desc,
- .num_pmc_entries = PFM_AMD_NUM_PMCS,
- .num_pmd_entries = PFM_AMD_NUM_PMDS,
- .probe_pmu = pfm_amd64_probe_pmu,
- .version = "1.2",
- .arch_info = &pfm_amd64_pmu_info,

```

