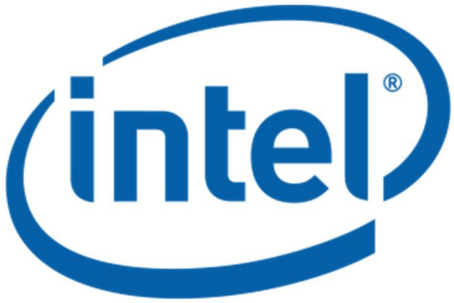

Intel® Xeon Phi™ Coprocessor (codename: Knights Corner) Performance Monitoring Units

Revision: 1.01
Last Modified: July 10, 2012
Document Number: 327357-001

Table of Contents

Table of Contents **4**

1 Intel® Xeon Phi™ Coprocessor Performance Monitoring Units **5**

1.1 Overview 5

1.2 Overview of Core Performance Monitoring Units (PMU) and Events 5

1.3 Performance Monitoring Programming Methodology 6

1.3.1 Expected Sampling Methodologies 6

1.3.2 Ring 0 vs. Ring 3 Programmability in the Core PMUs 7

1.3.3 Production vs. Instrumented Driver 7

1.4 Core PMU Programming 7

1.4.1 Basic Programming 8

1.4.2 Core PMU Instructions 9

1.4.3 Core PMU Control Registers 10

1.4.4 Warm Reset/INIT Behavior 20

1 Intel® Xeon Phi™ Coprocessor Performance Monitoring Units

1.1 Overview

Many of the challenges of complex application performance tuning arise from the fact that developers have very limited (or no) insight into the architectural specifics of how their code runs on a given silicon architecture. This is especially challenging for high performance computing applications where a product's success is directly determined by the quality of software performance tuning in both the driver and in the applications. Modern software performance tuning requires implementing detailed performance events and counters in hardware. Developer-friendly performance analysis tools (e.g. Intel® VTune™ Amplifier XE, etc.) can then configure and access these hardware counters to deliver accurate and sophisticated performance analysis of applications to the developer.

1.2 Overview of Core Performance Monitoring Units (PMU) and Events

The foundation for the Intel® Xeon Phi™ coprocessor core PMU is the PMU from the original Intel® Pentium® processor (aka P54C). Most of the forty-two performance events that were available in the original Intel® Pentium® processor are also available on the Intel® Xeon Phi™ coprocessor. The core PMU has been upgraded to an Intel® Pentium® Pro processor-like ("P6-style") programming interface. Below is a summary of the upgrades to the core PMU:

1. Array of Cores/PMUs: Each physical Intel® Xeon Phi™ coprocessor core has an independently-programmable core PMU. Thus, there is an "array of core PMUs" that correspond 1-to-1 to the array of physical Intel® Xeon Phi™ coprocessor x86 cores.

2. 4-Way Threaded: Each Intel® Xeon Phi™ coprocessor core is able to process 4 threads concurrently.

3. 2 Per-Thread Counters per Core: Each core PMU has 2 counters per thread instead of just two per core. Events with many-thread or no-thread context will also increment the counters.

4. P6-Style PMU Selection and Control: The programming interface to the core PMU has been upgraded to mimic the Intel® Pentium® Pro processor (P6) interface, i.e. "P6-style".

5. New Core PMU Instructions: Several instructions have been added to the Intel® Xeon Phi™ coprocessor instruction set to enhance performance monitoring capability for application-level code, i.e. Ring 3 code.

6. New Core PMU Events: Many new performance monitoring events have been added to provide insight into the new functional units, e.g. vector-processing unit (VPU), etc., available in the Intel® Xeon Phi™ coprocessor.

Each of these changes is discussed in detail in the rest of this document.

1.3 Performance Monitoring Programming Methodology

1.3.1 Expected Sampling Methodologies

1.3.1.1 Manual Instrumentation

In conjunction with a performance counter configuration tool (Ring 0), developers can manually instrument critical sections of their code with the new Ring 3 performance monitoring instructions, e.g. RDTSC, RDPMC, SPFLT, to obtain detailed performance counts. Note that the Intel® Xeon Phi™ coprocessor PMUs cannot be programmed at Ring 3. A (Ring 0) configuration tool will be built to program the core PMUs. Developers must use this tool to configure which hardware events they want to monitor.

1.3.1.2 Time-Based Sampling

It is expected that performance monitoring tools like Intel® VTune™ Amplifier XE will be built on a software stack that has Ring 0 access to the Intel® Xeon Phi™ coprocessor micro-operating system. These tools will interrupt the processor at a regular sampling interval to collect event counts and to reset the PMU counters if necessary.

1.3.1.3 Event-Based Sampling

Performance monitoring tools such as Intel® VTune™ Amplifier XE sometimes use an event-based mechanism for collecting performance monitoring information. Typically, these tools rely upon the overflow interrupt mechanism. More specifically, the tool will reset the PMU counters to values that are a “predictable distance away from overflowing” and will thus generate overflow interrupts at a desired beat rate. Because the event occurrence will generate an overflow interrupt, the entire state of the machine can be sampled at the time of the event to statistically characterize the machine state.

1.3.2 Ring 0 vs. Ring 3 Programmability in the Core PMUs

In the original Intel® Pentium® processor, access to performance counters was limited to Ring 0 drivers meaning that a front-end tool such as Intel® VTune™ Amplifier XE was required for end users. However, later architectures starting with the Intel® Pentium® Pro processor (P6) allowed Ring 3 application-level code to read the counter values via special instructions, e.g. RDPMC (Read Performance Monitor Counter). PMU configuration was still controlled by from Ring 0, but optimization-focused developers could now instrument code segments with RDPMC to directly retrieve performance information without going through the driver.

The core PMUs in the Intel® Xeon Phi™ coprocessor follows a very similar methodology to the P6: Intel® Xeon Phi™ coprocessor core PMUs provide the RDPMC instruction for Ring 3 application-level code to query its core performance counters. Configuration of those counters still requires a Ring 0-based interface. In addition, the new thread-based performance filtering capability can be controlled by application-level code via the new SPFLT instruction.

1.3.3 Production vs. Instrumented Driver

In general, access to the core PMUs is expected to be provided and accessible in the production driver stack including the production micro-operating system.

1.4 Core PMU Programming

Intel® Xeon Phi™ coprocessor core PMU provides 2 per-thread general purpose event counters. All of these counters are 40-bit in their precision. The counters may be configured to trigger Performance Monitoring Interrupts (PMI's) when event counts exceed the 40-bit representation. Event selection and each general propose counter's behavior is configured by a corresponding P6-style Performance Event Selection Register.

The model-specific registers (MSRs) to program core PMU operation can be read and written using the RDMSR and WRMSR instructions from Ring 0, respectively. All of these features and their MSRs are discussed in this section. In addition, Ring 3 access to the time-stamp counter, performance counters, and the performance filter mask register are provided via new instructions (also described in this section).

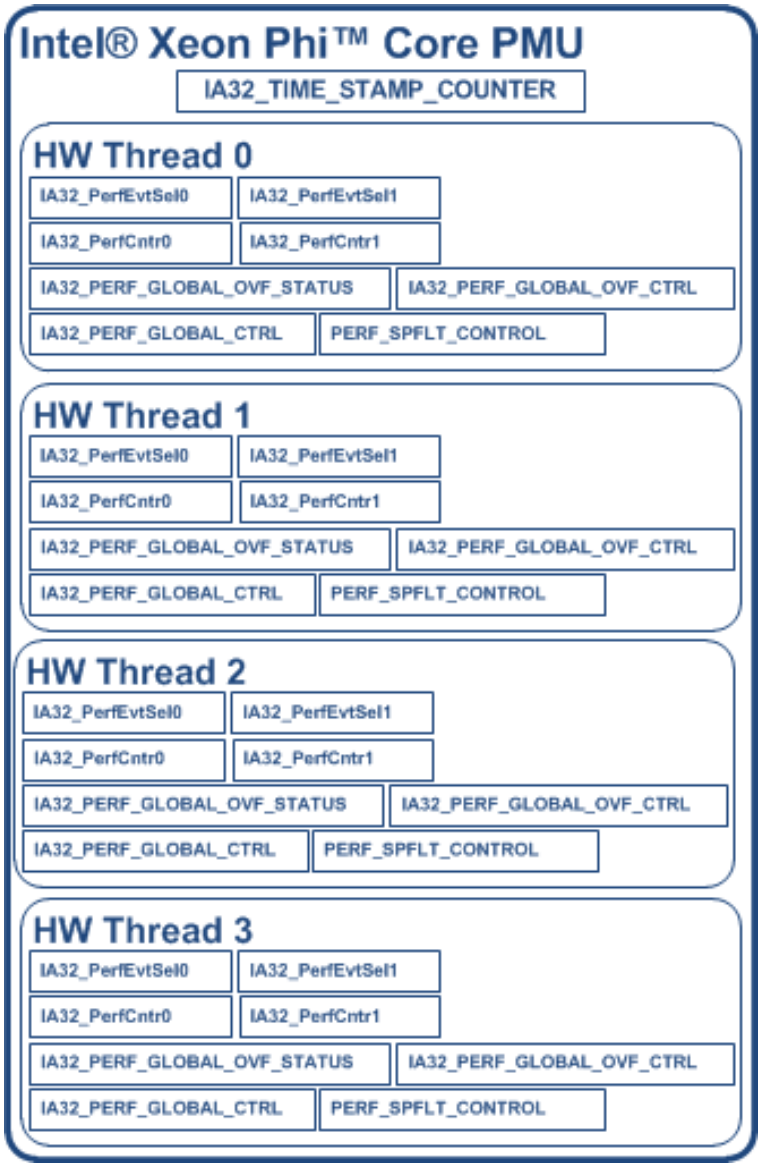

Figure 1.1: Core PMU Architecture Diagram

1.4.1 Basic Programming

Enabling the core PMU for event monitoring is relatively simple. The following registers must be configured. Once the logical AND of respective enable bits in the event selection and the master PMU registers is equal to one, the event monitoring hardware will begin filtering configured events into the counters.

- IA32_PerfEvtSelX and corresponding counters initialized
- IA32_PERF_GLOBAL_CTRL

1.4.2 Core PMU Instructions

The table below lists the instructions used by Ring 0 and Ring 3 code to control and query the core PMU as it applies to the running thread.

Table 1-1: Core PMU Instructions

Instruction Name	Description	Privilege Mode (CPL)	Thread-Specific	Input	Output
RDMSR	Read model specific register. Used by Ring 0 code to read any core PMU register.	Ring 0	Yes	ECX: Address of MSR	EDX:EAX = 64-bit MSR value
WRMSR	Write model specific register. Used by Ring 0 code to write any core PMU register	Ring 0	Yes	EDX:EAX = 64-bit MSR value ECX: Address of MSR	None
RDTSC	Read time-stamp counter. Reads the current time-stamp counter value.	Ring 0-3	No	None	EDX:EAX = 64-bit time-stamp value
RDPMC	Read performance monitoring counter. Reads the counts of any of the performance monitoring counters, including the PMU filtered counters.	Ring 0-3	Yes	ECX: Counter # 0x0: IA32_PerfCntr0 0x1: IA32_PerfCntr1	EDX:EAX = Zero-extended 40-bit counter value
SPFLT	Set user preference flag to indicate counter enable/disable.	Ring 0-3	Yes	Any GPR[0]: 0x0: Clear (disable) 0x1: Set (enable)	Set/clear USER_PREF bit in PERF_SPFLT_CONTROL.

The registers RDMSR, WRMSR, RDTSC, and RDPMC are well-documented (Intel® 64 and IA-32 Architectures Software Developer Manuals). The only Intel® Xeon Phi™ coprocessor-specific notes are that RDTSC has been enhanced to execute in 4-5 clock cycles.

SPFLT is unique because it allows software threads fine-grained control in enabling/disabling the performance counters. The anticipated usage model for this instruction is for instrumented code to enable/disable counters around desired portions of code. Note that software can only specify its preference for enabling/disabling counters and does not have control over which specific counters are affected (this behavior supports virtualization).

1.4.3 Core PMU Control Registers

The table below lists the model-specific registers used to program the operation of the core PMU.

Table 1-2: Core PMU Control Registers

Register Address		Name	Description	Threaded?	Width
Hex	Dec				
0x10	16	IA32_TIME_STAMP_COUNTER	"Time-Stamp Counter"	No	64
0x20	32	IA32_PerfCntr0	"Events Counted", core PMU counter 0	Yes	40
0x21	33	IA32_PerfCntr1	"Events Counted", core PMU counter 1	Yes	40
0x28	40	IA32_PerfEvtSel0	Performance Event Selection and configuration register for IA32_PerfCntr0.	Yes	32
0x29	41	IA32_PerfEvtSel1	Performance Event Selection and configuration register for IA32_PerfCntr1.	Yes	32
0x2C	44	PERF_SPFLT_CONTROL	"SPFLT Control Register" This MSR controls the effect of the SPFLT instruction and whether it will allow software fine-grained control to enable/disable IA32_PerfCntrN.	Yes	64
0x2D	45	IA32_PERF_GLOBAL_STATUS	"Counter Overflow Status" This read-only MSR displays the overflow status of all the counters. Each bit is implemented as a sticky bit, set by a counter overflow.	Yes	32
0x2E	46	IA32_PERF_GLOBAL_OVF_CTRL	"Counter Overflow Control" This write-only MSR clears the overflow indications in the Counter Overflow Status register. For each bit that is set, the corresponding overflow status is cleared.	Yes	32
0x2F	47	IA32_PERF_GLOBAL_CTRL	"Master PMU Enable" Global PMU enable / disable. When these bits are set, the core PMU is permitted to count events as configured by each of the Performance Event Selection registers (which can each be independently enabled or disabled). When these bits are cleared, performance monitoring is disabled. The operation of the Time-Stamp Counter is not affected by this register.	Yes	32

1.4.3.1 IA32_TIME_STAMP_COUNTER

Name: Time-Stamp Counter
Address: 0x10
Default Value: 0x0000000000000000
Normal Access: RW

Size: 64 bits
Threaded: No
Additional Notes: The time-stamp counter increments at core clock rate and is not appropriate for wall-clock use. There is only a single TSC per core, which is shared by all threads.

Table 1-3: IA32_TIME_STAMP_COUNTER Register Layout

Bit	Description
63:0	**Time-Stamp Counter**: Number of core clock cycles since reset

1.4.3.2 IA32_PerfCntr0/1

Name: Performance Events Counted
Address: 0x20 (0), 0x21 (1)
Default Value: 0x0000000000
Normal Access: RW
Size: 40 bits
Threaded: Yes
Additional Notes: Each of the 40-bit counters increments when a specified hardware event occurs and the counter is enabled and allowed to count the hardware event. Reads and writes to this register set all 40 bits which differs from the behavior of some IA32 processors which only write the lower 32 bits and sign extend the upper 8 bits.

Table 1-4: IA32_PerfCntr0/1 Register Layout

Bit	Description
63:40	**Reserved**
39:0	**Events Counted**: Number of events counted as specified by the associated performance event selection register.

1.4.3.3 IA32_PerfEvtSel0/1

Name: Performance Event Selection Register
Address: 0x28 (0), 0x29 (1)
Default Value: 0x00000000
Normal Access: RW
Size: 32 bits
Threaded: Yes
Additional Notes: None

Table 1-5: IA32_PerfEvtSel0/1 Register Layout

Bit	Description
31:24	**Counter mask (CMASK):** When nonzero, the processor compares this mask to the number of events counted during a single cycle. If the event count is greater than or equal to this mask, the counter is incremented by one. Otherwise, the counter is not incremented. This mask can be used to count events only if multiple occurrences happen per clock (for example, two or more instructions retired per clock). If the counter-mask field is 0, then the counter is incremented each cycle by the number of events that occurred that cycle.
23	**Invert (INV)**: Inverts the result of the counter-mask comparison when set, so that both greater-than-or-equal-to and less-than comparisons can be made.
22	**Enable Counter (EN)**: When set, performance counting is enabled if the corresponding bit in IA32_PERF_GLOBAL_CTRL bit is also set.
21	**Thread Count Mode:** When cleared, count events that match this thread's context. When set, count events that match any thread context. Events which have no thread-specific context will always be counted regardless of Thread Count Mode.
20	**APIC Interrupt Enable (INT)**: When set, the processor generates an exception through its local APIC on counter overflow.
19	**Reserved**
18	**Edge Detect (E)**: Enables (when set) edge detection of events. The processor counts the number of deasserted-to-asserted transitions of any condition that can be expressed by the other fields. The mechanism is limited in that it does not permit back-to-back assertions to be distinguished. This mechanism allows software to measure not only the fraction of time spent in a particular state, but also the average length of time spent in such a state (for example, the time spent waiting for an interrupt to be serviced).
17	**Operating System Mode (OS)**: Enables counting events when the processor is operating at Ring 0. This flag can be used in conjunction with the USR flag to allow event counting at any privilege level.
16	**User Mode (USR)**: Enables counting events when the processor is operating at Ring 1, 2 or 3. This flag can be used in conjunction with the OS flag to allow event counting at any privilege level.
15:8	**Unit Mask (UMASK)**: Further qualifies the event logic unit selected in the event select field to detect a specific architectural condition. For example, for some cache events, the mask is used as a MESI-protocol qualified of cache states.
7:0	**Event Select**: This 8-bit field is used to encode the desired hardware event. Illegal event encodings do not cause any hardware exceptions or faults and also do not increment the counters.

Table 1-6: List of Events

FUB	UMASK	Event Code	Mnemonic Event Name	Description
P54C	0x00	0x00	DATA_READ	Number of successful memory data reads committed by the K-unit (L1). Cache accesses resulting from prefetch instructions are included for A0 stepping.
P54C	0x00	0x01	DATA_WRITE	Number of successful memory data writes committed by the K-unit (L1). Streaming stores (hit/miss L1), cacheable write partials, and UC promotions are all included.
P54C	0x00	0x02	DATA_PAGE_WALK	Number of data page walks

P54C	0x00	0x03	DATA_READ_MISS	Number of memory read accesses that miss the internal data cache whether or not the access is cacheable or noncacheable. Cache accesses resulting from prefetch instructions are not included.
P54C	0x00	0x04	DATA_WRITE_MISS	Number of memory write accesses that miss the internal data cache whether or not the access is cacheable. Non-cacheable misses are not included.
P54C	0x00	0x06	DATA_CACHE_LINES_WRITTEN_BACK	Number of dirty lines (all) that are written back, regardless of the cause
P54C	0x00	0x09	MEMORY_ACCESSES_IN_BOTH_PIPES	Number of data memory reads or writes that are paired in both pipes of the pipeline
P54C	0x00	0x0A	BANK_CONFLICTS	Number of actual bank conflicts
P54C	0x00	0x0C	CODE_READ	Number of instruction reads; whether the read is cacheable or noncacheable
P54C	0x00	0x0D	CODE_PAGE_WALK	Number of code page walks
P54C	0x00	0x0E	CODE_CACHE_MISS	Number of instruction reads that miss the internal code cache; whether the read is cacheable or noncacheable
P54C	0x00	0x11	L1_DATA_PF1	Number of data vprefetch0 requests seen by the L1.
P54C	0x00	0x12	BRANCHES	Number of taken and not taken branches, including: conditional branches, jumps, calls, returns, software interrupts, and interrupt returns
P54C	0x00	0x15	PIPELINE_FLUSHES	Number of pipeline flushes that occur Pipeline flushes are caused by BTB misses on taken branches, mispredictions, exceptions, interrupts, and some segment descriptor loads.
P54C	0x00	0x16	INSTRUCTIONS_EXECUTED	Number of instructions executed (up to two per clock)
P54C	0x00	0x17	INSTRUCTIONS_EXECUTED_V_PIPE	Number of instructions executed in the V_pipe The event indicates the number of instructions that were paired.
P54C	0x00	0x1C	L1_DATA_PF1_MISS	Number of data vprefetch0 requests seen by the L1 which missed L1. Does not include vprefetch1 requests which are counted in L1_DATA_PF1_DROP.
P54C	0x00	0x1E	L1_DATA_PF1_DROP	Number of data vprefetch0 requests seen by the L1 which were dropped for any reason. A vprefetch0 can be dropped if the requested address matches another in-flight request or if it has a UC memtype.
P54C	0x00	0x1F	PIPELINE_AGI_STALLS	Number of address generation interlock (AGI) stalls. An AGI occurring in both the U- and V- pipelines in the same clock signals this event twice.
P54C	0x00	0x20	L1_DATA_HIT_INFLIGHT_PF1	Number of data requests which hit an in-flight vprefetch0. The in-flight vprefetch0 was not necessarily issued from the same

				thread as the data request.
P54C	0x00	0x21	PIPELINE_SG_AGI_STALLS	Number of address generation interlock (AGI) stalls due to vscatter* and vgather* instructions.
P54C	0x00	0x28	DATA_READ_OR_WRITE	Number of memory data reads and/or writes (internal data cache hit and miss combined). Read cache accesses resulting from prefetch instructions are included for A0 stepping.
P54C	0x00	0x29	DATA_READ_MISS_OR_WRITE_MISS	Number of memory read and/or write accesses that miss the internal data cache, whether or not the access is cacheable or noncacheable
P54C	0x00	0x2A	CPU_CLK_UNHALTED	Number of cycles during which the processor is not halted.
P54C	0x00	0x2B	BRANCHES_MISPREDICTED	Number of branch mispredictions that occurred on BTB hits. BTB misses are not considered branch mispredicts because no prediction exists for them yet.
P54C	0x00	0x2C	MICROCODE_CYCLES	The number of cycles microcode is executing. While microcode is executing, all other threads are stalled.
P54C	0x00	0x2D	FE_STALLED	Number of cycles where the front-end could not advance. Any multi-cycle instructions which delay pipeline advance and apply backpressure to the front-end will be included, e.g. read-modify-write instructions. Includes cycles when the front-end did not have any instructions to issue.
P54C	0x00	0x2E	EXEC_STAGE_CYCLES	Number of E-stage cycles that were successfully completed. Includes cycles generated by multi-cycle E-stage instructions. For instructions destined for the FPU or VPU pipelines, this event only counts occupancy in the integer E-stage.
P54C	0x00	0x37	L1_DATA_PF2	Number of data vprefetch1 requests seen by the L1. This is not necessarily the same number as seen by the L2 because this count includes requests that are dropped by the core. A vprefetch1 can be dropped by the core if the requested address matches another in-flight request or if it has a UC memtype.
P54C	0x00	0x38	L2_DATA_PF1_MISS	Number of data vprefetch0 requests seen by the L2 which missed L2.
P54C	0x00	0x3A	LONG_DATA_PAGE_WALK	Number of "long" data page walks, i.e. page walks that also missed the L2 uTLB. Subset of DATA_PAGE_WALK event
P54C	0x00	0x3B	LONG_CODE_PAGE_WALK	Number of "long" code page walks, i.e. page walks that also missed the L2 uTLB. Subset of DATA_CODE_WALK event
CRI	0x10	0xC8	L2_READ_HIT_E	L2 Read Hit E State, may include

				prefetches on A0 stepping.
CRI	0x10	0xC9	L2_READ_HIT_M	L2 Read Hit M State
CRI	0x10	0xCA	L2_READ_HIT_S	L2 Read Hit S State
CRI	0x10	0xCB	L2_READ_MISS	L2 Read Misses. Prefetch and demand requests to the same address will produce double counting.
CRI	0x10	0xCC	L2_WRITE_HIT	L2 Write HIT, may undercount on A0 stepping.
CRI	0x10	0xD7	L2_VICTIM_REQ_WITH_DATA	L2 received a victim request and responded with data
CRI	0x10	0xE3	SNP_HITM_BUNIT	Snoop HITM in BUNIT
CRI	0x10	0xE6	SNP_HIT_L2	Snoop HIT in L2
CRI	0x10	0xE7	SNP_HITM_L2	Snoop HITM in L2
CRI	0x10	0xF0	L2_CODE_READ_MISS_CACHE_FILL	Number of code read accesses that missed the L2 cache and were satisfied by another L2 cache. Can include promoted read misses that started as DATA accesses.
CRI	0x10	0xF1	L2_DATA_READ_MISS_CACHE_FILL	Number of data read accesses that missed the L2 cache and were satisfied by another L2 cache. Can include promoted read misses that started as CODE accesses.
CRI	0x10	0xF2	L2_DATA_WRITE_MISS_CACHE_FILL	Number of data write (RFO) accesses that missed the L2 cache and were satisfied by another L2 cache.
CRI	0x10	0xF5	L2_CODE_READ_MISS_MEM_FILL	Number of code read accesses that missed the L2 cache and were satisfied by main memory. Can include promoted read misses that started as DATA accesses.
CRI	0x10	0xF6	L2_DATA_READ_MISS_MEM_FILL	Number of data read accesses that missed the L2 cache and were satisfied by main memory. Can include promoted read misses that started as CODE accesses.
CRI	0x10	0xF7	L2_DATA_WRITE_MISS_MEM_FILL	Number of data write (RFO) accesses that missed the L2 cache and were satisfied by main memory.
CRI	0x10	0xFC	L2_DATA_PF2	Number of data vprefetch1 requests seen by the L2. Only counts vprefetch1 hits on A0 stepping.
CRI	0x10	0xFD	L2_DATA_PF2_DROP	Number of data vprefetch1 requests seen by the L2 which were dropped for any reason.
CRI	0x10	0xFE	L2_DATA_PF2_MISS	Number of data vprefetch1 requests seen by the L2 which missed L2. Does not include vprefetch2 requests which are counted in L2_DATA_PF2_DROP.
CRI	0x10	0xFF	L2_DATA_HIT_INFLIGHT_PF2	Number of data requests which hit an in-flight vprefetch1. The in-flight vprefetch1 was not necessarily issued from the same thread as the data request.
VPU	0x20	0x00	VPU_DATA_READ	Number of read transactions that were issued. In general each read transaction will read one 64B cacheline. If there are alignment issues, then reads against

				multiple cache lines will each be counted individually.
VPU	0x20	0x01	VPU_DATA_WRITE	Number of write transactions that were issued. . In general each write transaction will write one 64B cacheline. If there are alignment issues, then write against multiple cache lines will each be counted individually.
VPU	0x20	0x03	VPU_DATA_READ_MISS	VPU L1 data cache readmiss. Counts the number of occurrences.
VPU	0x20	0x04	VPU_DATA_WRITE_MISS	VPU L1 data cache write miss. Counts the number of occurrences.
VPU	0x20	0x05	VPU_STALL_REG	VPU stall on Register Dependency. Counts the number of occurrences. Dependencies will include RAW, WAW, WAR.
VPU	0x20	0x16	VPU_INSTRUCTIONS_EXECUTED	Counts the number of VPU instructions executed in both u- and v-pipes.
VPU	0x20	0x17	VPU_INSTRUCTIONS_EXECUTED_V_PIPE	Counts the number of VPU instructions that paired and executed in the v-pipe.
VPU	0x20	0x18	VPU_ELEMENTS_ACTIVE	Counts the cumulative number of elements active (via mask) for VPU instructions issued.

1.4.3.4 PERF_SPFLT_CONTROL

Name: SPFLT Control Register
Address: 0x2C
Default Value: 0x00000000_00000000
Normal Access: RW
Size: 64 bits
Threaded: Yes
Additional Notes: None

Table 1-7: PERF_SPFLT Register Layout

Bit	Description
63	**User Preference:** Software-controlled preference bit to enable/disable counters. When this bit is 0, software is declaring preference to disable counters. When this bit is 1, software is declaring preference to enable counters. Note that this bit only affects counters that have specific SPFLT Enable Counter N bit set also, i.e., logical AND of the User Preference and SPFLT Counter Enable N bits.
62:2	**Reserved**
1	**SPFLT Enable Counter 1**: When set, allows SPFLT to count events for IA32_PerfCntr1. Used in combination with the User Preference bit. This bit is controlled by the operating system.
0	**SPFLT Enable Counter 0**: When set, allows SPFLT to count events for IA32_PerfCntr0. Used in combination with the User Preference bit. This bit is controlled by the operating system.

1.4.3.5 IA32_PERF_GLOBAL_STATUS

Name: Performance Counter Overflow Status
Address: 0x2D
Default Value: 0x00000000
Normal Access: RO
Size: 32 bits
Threaded: Yes
Additional Notes: Each bit is implemented as a sticky bit, set by a counter overflow. This register will update even if APIC interrupts are not enabled. The intended use of this register by interrupt service routines is to see which counters have overflowed.

Table 1-8: IA32_PERF_GLOBAL_STATUS Register Layout

Bit	Description
31:2	**Reserved**
1	**PerfCntr1**: Counter 1 caused an overflow.
0	**PerfCntr0**: Counter 0 caused an overflow.

1.4.3.6 IA32_PERF_GLOBAL_OVF_CONTROL

Name: Performance Counter Overflow Status
Address: 0x2E
Default Value: 0x00000000
Normal Access: WO
Size: 32 bits
Threaded: Yes
Additional Notes: None

Table 1-9: IA32_PERF_GLOBAL_OVF_CONTROL Register Layout

Bit	Description
63:2	**Reserved**
1	**PerfCntr1**: Clear overflow caused by counter 1.
0	**PerfCntr0**: Clear overflow caused by counter 0.

1.4.3.7 IA32_PERF_GLOBAL_CTRL

Name: Global Performance Counter Control
Address: 0x2F
Default Value: 0x00000000
Normal Access: RW

Size: 32 bits
Threaded: Yes
Additional Notes: These global bits are used in conjunction with each individual IA32_PerfEvtSelx register's enable bit.

Table 1-10: IA32_PERF_GLOBAL_CTRL Register Layout

Bit	Description
31:2	**Reserved**
1	**Enable Performance Monitoring via IA32_PerfEvtSel1**: 0: Counter 1 is disabled, 1: Counter 1 is enabled.
0	**Enable Performance Monitoring via IA32_PerfEvtSel0**: 0: Counter 0 is disabled, 1: Counter 0 is enabled.

1.4.3.8 Handling PMIs in the Core PMU

With 40-bit counters, events that occur once per core clock cycle will take approximately 18 minutes to overflow assuming a clock frequency of 1.0GHz. Virtually all configurable hardware events occur much less frequently than this, so counter overflow should not be a significant problem. However, for sampling usage models, where counters are artificially set to a near-overflow condition, overflow interrupts may be a much more common occurrence. The counters can be configured to deliver an interrupt to the local APIC when an overflow condition is reached. The overflow status of all the counters is mirrored in the read-only IA32_PERF_GLOBAL_STATUS status register. Each thread has its own local APIC and the core PMU routes the interrupt to the APIC whose thread generated the overflow condition. For events with no thread-specific context, events are routed to whichever thread happens to be in the execution stage at that time. Once the local APIC receives the signal from the core PMU, it references its local vector table (LVT) entry for performance counter overflow events (offset 0x340), which directs it to the entry within the Interrupt Descriptor Table (IDT). The IDT entry should be instrumented by the EMON driver, which will be responsible for executing instructions to clear the overflow condition. The corresponding interrupt service routine should unmask interrupts, write into the IA32_PERF_GLOBAL_OVF_CONTROL to clear the interrupt, and signal end-of-interrupt (EOI).

1.4.3.9 Multiple / Rapid Counter Overflows

In rare instances, multiple counters may overflow simultaneously and/or in rapid succession, generating multiple and/or rapid performance monitoring interrupts (PMIs) to the local APIC. Since there is only a 1-bit interrupt signal between the core PMU and the threaded APIC, communicating multiple overflow interrupts is impossible. Multiple / rapid interrupt behavior is further complicated by APIC behavior introduced in the Intel® Pentium® 4 processor family to automatically mask the Performance Counter Register LVT entry whenever an overflow interrupt is received. An excerpt from the Software Developer's Manual (Intel® 64 and IA-32 Architectures Software Developer Manuals):

> "When the local APIC handles a performance-monitoring counter interrupt, it automatically sets the mask flag in the corresponding LVT entry. This flag will remain set until software clears it."

However, in combination with the IA32_PERF_GLOBAL_STATUS register and some software mechanisms in the interrupt service routine (ISR), the 1-bit interrupt signal is enough to handle these situations.

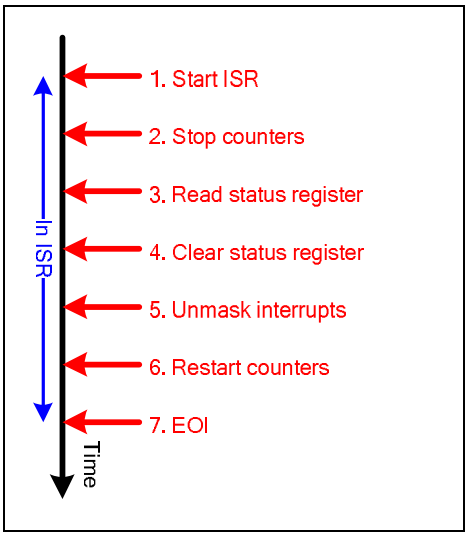

Figure 1.2: Sample Execution Timeline for Counter Overflow Interrupt Delivery

The figure above shows how a PMI is serviced by the ISR. By carefully following the detailed steps, dropped / delayed PMIs can be avoided even in situations when there are multiple / rapid interrupts.

The most important points are as follows:

- The 1-bit PMI interface to the APIC transitions from '0 to '1 whenever a counter needs servicing. The identities of the overflowed counters are stored in the overflow status register. So, it does not matter if multiple interrupts occur at the same time, nor does it matter if unique interrupts occur rapidly in succession. The first interrupt alerts the APIC and the rest of the interrupts are tracked in the overflow status register.

- The sole responsibility of the ISR is to clear the overflow status register to zero before unmasking interrupts. That is why the counters are stopped in step 2 to prevent additional PMIs. If the counters are not stopped in step 2, any PMIs after step 3 will not be cleared and hence the interrupt line to the APIC will remain '1. In order to request another interrupt to the APIC, the line must transition from '0 to '1 while interrupts are unmasked.

1.4.4 Warm Reset/INIT Behavior

From the core PMU perspective, a warm reset is identical to a cold reset. In other words, all the control register and counter state is reset to zero. An INIT does not affect the core PMU control register state or counters.