Intel x86 Architecture PMC

Mina Gong
Electrical and Computer Engineering Department
University of California, Davis
mneronde@ucdavis.edu

1. Download The Manual

- Intel 64 and IA-32 Architectures Software Developer's Maunal
  - Link: https://software.intel.com/en-us/articles/intel-sdm
- Environment: Intel x86 architecture 3rd generation called Ivy Bridge
  - Ivy Bridge uses the same description with Sandy Bridge in the manual
  - command 'lscpu' in the terminal of Linux provides the information of the processor

2. Prepare Linux Environment

- The Lab Computer (Host PC) has installed with Ubuntu 18.04.4 distro
- Ubuntu 18.04.4 kernel version: Linux Kernel v.5.3
- Download Linux kernel code: https://www.kernel.org/

  - Important: download the kernel code version matching or lower than the host PC kernel version

  - e.g. For Linux kernel v.5.3 of the host pc, downloaded 4.19.120 (Above 5.3 version doesn't work)

  - after downloading in any directory in Linux, upzip the file by using the command 'tar –xvf filename'

mainline:	5.7-rc3	2020-04-26	[tarball]		[patch]	[inc. patch]	[view diff]	[browse]
stable:	5.6.10	2020-05-02	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse] [changelog]
stable:	5.5.19 [EOL]	2020-04-21	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse] [changelog]
longterm:	5.4.38	2020-05-02	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse] [changelog]
longterm:	4.19.120	2020-05-02	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse] [changelog]
longterm:	4.14.178	2020-05-02	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse] [changelog]
longterm:	4.9.221	2020-05-02	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse] [changelog]
longterm:	4.4.221	2020-05-02	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse] [changelog]
longterm:	3.16.83	2020-04-28	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse] [changelog]
linux-next:	next-20200501	2020-05-01						[browse]

2. Prepare Linux Environment (Cont)

- Why installing kernel code from kernel.org with the version that is lower or same?

  - Versions above have different code (Table 1)

Ubuntu Kernel v5.3	Kernel Code v5.5	Kernel code 4.9
#define INTEL_FAM6_SANDYBRIDGE 0x2A	#define INTEL_FAM6_SANDYBRIDGE 0x2A	#define INTEL_FAM6_SANDYBRIDGE 0x2A
#define INTEL_FAM6_SANDYBRIDGE_X 0x2D	#define INTEL_FAM6_SANDYBRIDGE_X 0x2D	#define INTEL_FAM6_SANDYBRIDGE_X 0x2D
#define INTEL_FAM6_IVYBRIDGE 0x3A	#define INTEL_FAM6_IVYBRIDGE 0x3A	#define INTEL_FAM6_IVYBRIDGE 0x3A
#define INTEL_FAM6_IVYBRIDGE_X 0x3E	#define INTEL_FAM6_IVYBRIDGE_X 0x3E	#define INTEL_FAM6_IVYBRIDGE_X 0x3E
#define INTEL_FAM6_HASWELL_CORE 0x3C	#define INTEL_FAM6_HASWELL 0x3C	#define INTEL_FAM6_HASWELL_CORE 0x3C
#define INTEL_FAM6_HASWELL_X 0x3F	#define INTEL_FAM6_HASWELL_X 0x3F	#define INTEL_FAM6_HASWELL_X 0x3F
#define INTEL_FAM6_HASWELL_ULT 0x45	#define INTEL_FAM6_HASWELL_L 0x45	#define INTEL_FAM6_HASWELL_ULT 0x45
#define INTEL_FAM6_HASWELL_GT3E 0x46	#define INTEL_FAM6_HASWELL_G 0x46	#define INTEL_FAM6_HASWELL_GT3E 0x46
#define INTEL_FAM6_BROADWELL_CORE 0x3D	#define INTEL_FAM6_BROADWELL 0x3D	#define INTEL_FAM6_BROADWELL_CORE 0x3D
#define INTEL_FAM6_BROADWELL_GT3E 0x47	#define INTEL_FAM6_BROADWELL_G 0x47	#define INTEL_FAM6_BROADWELL_GT3E 0x47
#define INTEL_FAM6_BROADWELL_X 0x4F	#define INTEL_FAM6_BROADWELL_X 0x4F	#define INTEL_FAM6_BROADWELL_X 0x4F
#define INTEL_FAM6_BROADWELL_XEON_D 0x56	#define INTEL_FAM6_BROADWELL_D 0x56	#define INTEL_FAM6_BROADWELL_XEON_D 0x56

<Table 1: Different code depending on the kernel version>

- When use command ‘grep,’ some results are not shown in the host kernel so that use grep to do a specific search in downloaded kernel code.

```
mina@mina-OptiPlex-3010:~/Downloads/linux-4.19.113$ sudo grep -r 'INTEL_FAM6_IVYBRIDGE' ./*
./arch/x86/events/intel/cstate.c:       X86_CSTATES_MODEL(INTEL_FAM6_IVYBRIDGE,   snb_cstates),
./arch/x86/events/intel/cstate.c:       X86_CSTATES_MODEL(INTEL_FAM6_IVYBRIDGE_X, snb_cstates),
./arch/x86/events/intel/core.c: case INTEL_FAM6_IVYBRIDGE:
./arch/x86/events/intel/core.c: case INTEL_FAM6_IVYBRIDGE_X:
./arch/x86/events/intel/core.c:         if (boot_cpu_data.x86_model == INTEL_FAM6_IVYBRIDGE_X)
./arch/x86/events/intel/uncore.c:       X86_UNCORE_MODEL_MATCH(INTEL_FAM6_IVYBRIDGE,      ivb_uncore_init),
./arch/x86/events/intel/uncore.c:       X86_UNCORE_MODEL_MATCH(INTEL_FAM6_IVYBRIDGE_X,    ivbep_uncore_init),
./arch/x86/events/intel/rapl.c: X86_RAPL_MODEL_MATCH(INTEL_FAM6_IVYBRIDGE,   snb_rapl_init),
./arch/x86/events/intel/rapl.c: X86_RAPL_MODEL_MATCH(INTEL_FAM6_IVYBRIDGE_X, snbep_rapl_init),
./arch/x86/events/msr.c:        case INTEL_FAM6_IVYBRIDGE:
./arch/x86/events/msr.c:        case INTEL_FAM6_IVYBRIDGE_X:
mina@mina-OptiPlex-3010:/usr/src/linux-headers-5.3.0-40-generic/arch/x86/events/intel$ sudo grep -r 'INTEL_FAM6_IVYBRIDGE' ./*
mina@mina-OptiPlex-3010:/usr/src/linux-headers-5.3.0-40-generic/arch/x86/events/intel$
```

3. Check How Many Performance Counters Available

- According to the Manual (p.692), Sandy bridge has 8 performance counters available

  - On the bottom contents of register figure, it says CPUID.0AH:EAX[15:8] reports a value of ‘8’, PMC4 – PMC7 are valid

- PMC0 – PMC3 are absolutely available, and details about PMC5 – PMC7 (if PMC present) is followed by the next slide

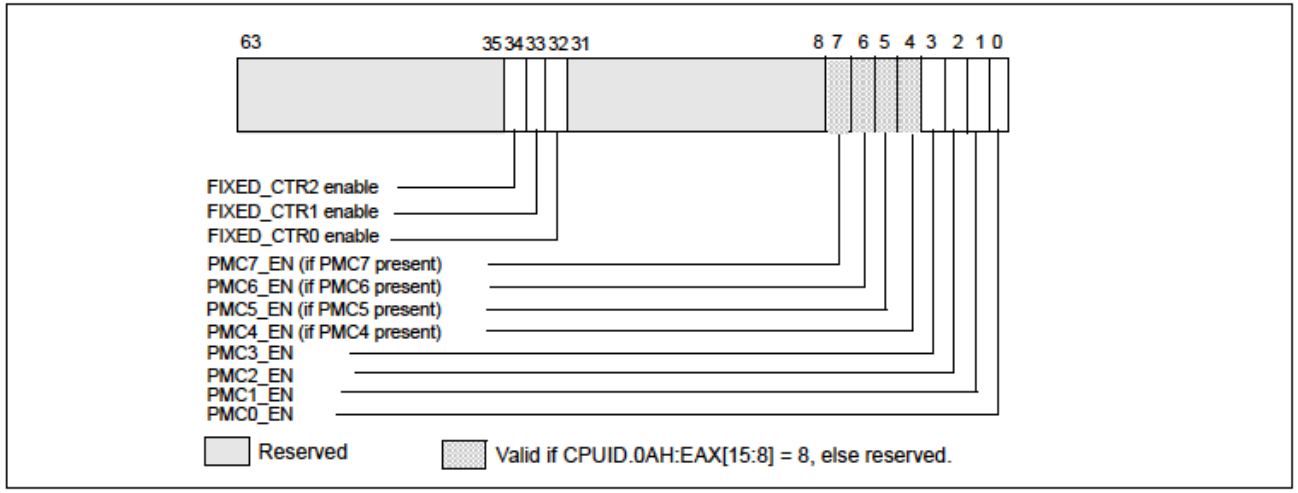

Figure 18-32. IA32_PERF_GLOBAL_CTRL MSR in Intel® Microarchitecture Code Name Sandy Bridge

Figure 18-15 depicts the layout of IA32_PERF_GLOBAL_CTRL MSR. The enable bits (PMC4_EN, PMC5_EN, PMC6_EN, PMC7_EN) corresponding to IA32_PMC4-IA32_PMC7 are valid only if CPUID.0AH:EAX[15:8] reports a value of ‘8’. If CPUID.0AH:EAX[15:8] = 4, attempts to set the invalid bits will cause #GP.

3. Check How Many Performance Counters Available (Cont)

- To find whether PMC5 – PMC7 are available, check CPUID.0AH:EAX[15:8] reports 8, which are composed of 2 bits of Reserved, 2 bits of Processor Type, and 4 bits of Family ID

Processor Version Information

EAX																											
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				Extended Family ID								Extended Model ID				Reserved		Processor Type		Family ID				Model				Stepping ID			

- In the terminal of Linux, type cpuid to see the information about EAX[15:8]

  - if the command is not installed, install it first

  - command: sudo apt-get install cpuid

```
[mina@mina-OptiPlex-3010:~/Downloads/linux-5.5.3/drivers/hellotest2$ cpuid                    ]
CPU 0:
   vendor_id = "GenuineIntel"
   version information (1/eax):
      processor type  = primary processor (0)
      family          = Intel Pentium Pro/II/III/Celeron/Core/Core 2/Atom, AMD Athlon/Duron, Cyrix
 M2, VIA C3 (6)
      model           = 0xa (10)
      stepping id     = 0x9 (9)
      extended family = 0x0 (0)
      extended model  = 0x3 (3)
```

3. Check How Many Performance Counters Available (Cont)

- Family ID, which is EAX[11:8], is 6 according to the command ‘cupid’
- Processor Type, which is EAX[13:12], is 0
- Reserved, which is EAX[15:14], can be 4 cases
    - Reserved can be 0,1,2,3 (bit [7:6] in the Table 2)
    - None of the cases could be 8 so that when running the PMCs, 4 of PMCs are available
    - Some of Ivy Bridge processor was able to read 8 PMCs
    - It could be sorely applicable to Sandy Bridge since the manual doesn’t have the section only for IvyBridge

	7	6	5	4	3	2	1	0	Decimal
1	0	0	0	0	0	1	1	0	6
2	0	1	0	0	0	1	1	0	70
3	1	0	0	0	0	1	1	0	134
4	1	1	0	0	0	1	1	0	198

<Table 2: Different code depending on the kernel version>

4. Source Code

- Source code is based on this manual: Intel 64 and IA-32 Architectures Software Developer's Manual, Volume3 (3A, 3B, 3C & 3D): System Programming Guide
- Source code has been attached with this instruction document
- For Sandy Bridge processor, addresses of registers starts from p.1435

4. Source Code (Cont)

1) Control Register 4 (Manual p.79)
   - Bit 8 of CR4 (Control Register 4) needs to be set to use RDPMC instructions
   - Two system calls we can use:
     a. __read_cr4()
     b. __write_cr4(value)
   - Read cr4 to keep other bits same as before (we need change only bit 8)
   - Use a bit-wise operation to change bit 8 of the value obtained from __read_cr4(), Write the value back to the register

```
1   static void set_pce(void *arg)
2   {
3       int to_val = (arg != 0);
4       unsigned int cr4_val;
5
6       cr4_val = __read_cr4();
7       //printk("cr4_val: %08x\n", cr4_val);
8       if(to_val) {
9           cr4_val |= X86_CR4_PCE;
10      } else {
11          cr4_val &= ~X86_CR4_PCE;
12      }
13      //printk("cr4_val: %08x\n", cr4_val);
14      __write_cr4(cr4_val);
15  }
```

4. Source Code (Cont)

2) Global Control Register (Manual p.692)

- To use PMCs, we need to modify PERF_GLOBAL_CTRL MSR
- [n:0] bits are PMC enable bits
  * N is dependent on available PMCs
- Method to check how many PMCs is in chapter 3 (5th slide)
- Perf_GLOBAL_CTRL MSR address is presented in p.1445
- Its address is 0x38F

18.9.1 Global Counter Control Facilities In Intel® Microarchitecture Code Name Sandy Bridge

The number of general-purpose performance counters visible to a logical processor can vary across Processors based on Intel microarchitecture code name Sandy Bridge. Software must use CPUID to determine the number performance counters/event select registers (See Section 18.2.1.1).

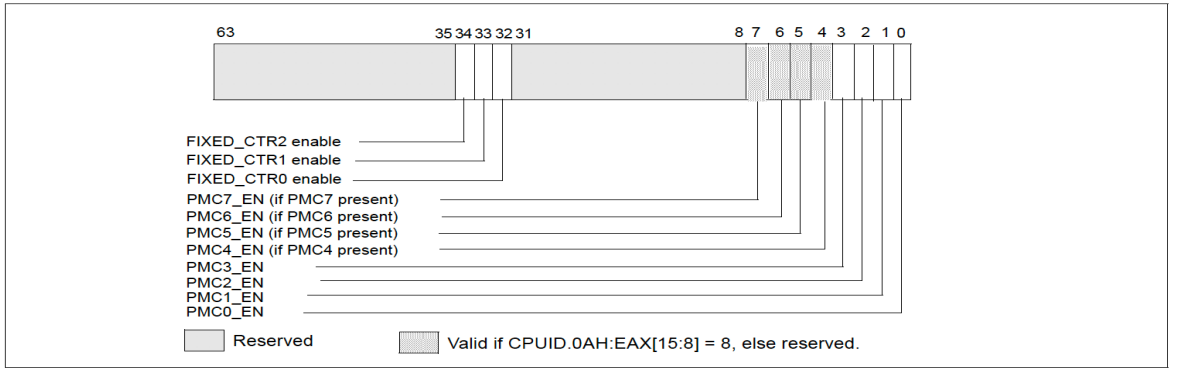

Figure 18-32. IA32_PERF_GLOBAL_CTRL MSR in Intel® Microarchitecture Code Name Sandy Bridge

Table 35-18. MSRs Supported by Intel® Processors based on Intel® microarchitecture code name Sandy Bridge (Contd.)

38FH	911	IA32_PERF_GLOBAL_CTRL	Thread	See Table 35-2. See Section 18.4.2, "Global Counter Control Facilities."
		0	Thread	**Set 1 to enable PMC0 to count**
		1	Thread	**Set 1 to enable PMC1 to count**
		2	Thread	**Set 1 to enable PMC2 to count**
		3	Thread	**Set 1 to enable PMC3 to count**
		4	Core	**Set 1 to enable PMC4 to count (if CPUID.0AH:EAX[15:8] > 4)**
		5	Core	**Set 1 to enable PMC5 to count (if CPUID.0AH:EAX[15:8] > 5)**
		6	Core	**Set 1 to enable PMC6 to count (if CPUID.0AH:EAX[15:8] > 6)**
		7	Core	**Set 1 to enable PMC7 to count (if CPUID.0AH:EAX[15:8] > 7)**
		31:8		Reserved.
		32	Thread	**Set 1 to enable FixedCtr0 to count**
		33	Thread	**Set 1 to enable FixedCtr1 to count**
		34	Thread	**Set 1 to enable FixedCtr2 to count**
		63:35		Reserved.

4. Source Code (Cont)

```
103         /*********************************************
104          * IA32_PERF_GLOBAL_CTRL setting
105          *********************************************/
106         msr = 0x38F;
107         rdmsr(msr, low, high);
108         //printk("##READ BEFORE SETTING## IA32_PERF_GLOBAL_CTRL 0x38F: [low]%08x, [high]%08x\n", low, high);
109
110         low |= 0xF;
111         wrmsr(msr, low, high);
112         //printk("##WRITE## IA32_PERF_GLOBAL_CTRL 0x38F: [low]%08x, [high]%08x\n", (low), (high));
113
114         rdmsr(msr, low, high);
115         //printk("##READ## IA32_PERF_GLOBAL_CTRL 0x38F: [low]%08x, [high]%08x\n", (low), (high));
```

2) Global Control Register (Cont)

- To keep the original value, read MSR value in the address 0x38F
- Depending on the number of PMCs, 0xFF (in case of 8 PMCs) or 0xF (in case of 4 PMCs) is applied to the value from previous step
- Double check the value is set by reading the register again

4. Source Code (Cont)

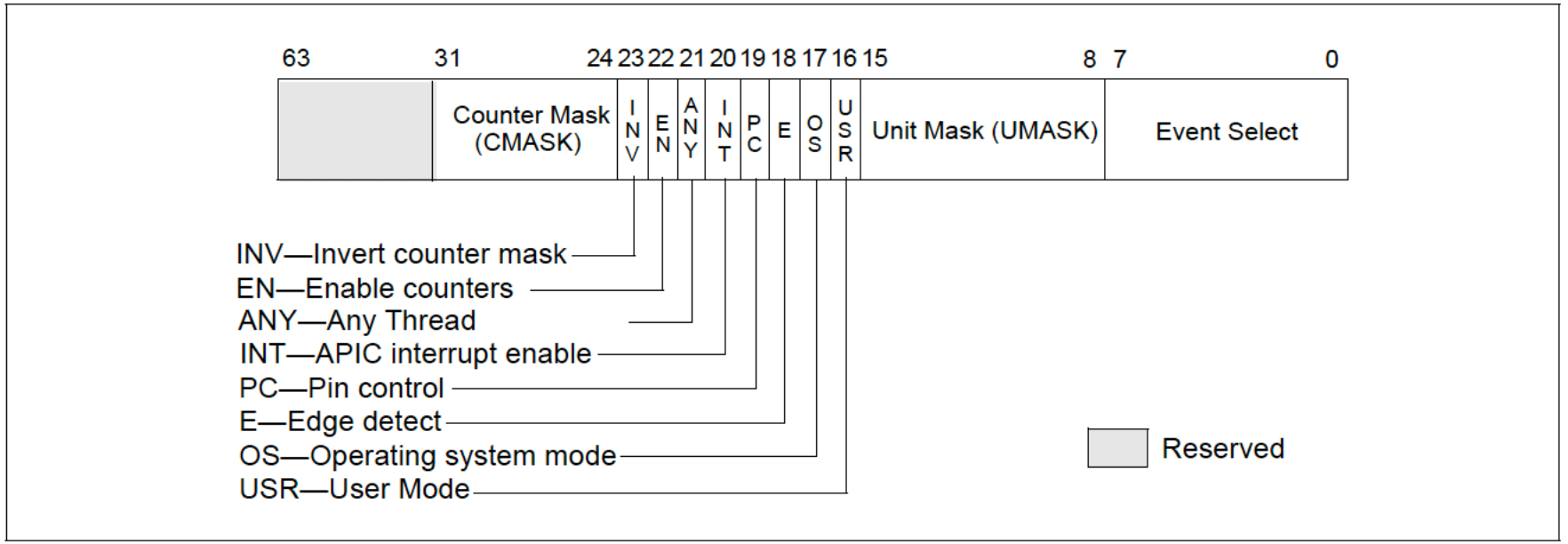

Figure 18-6. Layout of IA32_PERFEVTSELx MSRs Supporting Architectural Performance Monitoring Version 3

3) Event Register (Manual p.650)

- We need to set bit[23:16] to 0x43, which enables EN, OS, and USR bits, according to the Intel official document (Not the manual)
- No need to change bits other than bit 22 and bit[15:0]
- Unit Mask (UMASK) and Event Select [15:0]: Possible events are followed in the next slide

4. Source Code (Cont)

Table 19-1. Architectural Performance Events

Event Num.	Event Mask Name	Umask Value	Description
3CH	UnHalted Core Cycles	00H	Counts core clock cycles whenever the logical processor is in C0 state (not halted). The frequency of this event varies with state transitions in the core.
3CH	UnHalted Reference Cycles[1]	01H	Counts at a fixed frequency whenever the logical processor is in C0 state (not halted).
C0H	Instructions Retired	00H	Counts when the last uop of an instruction retires.
2EH	LLC Reference	4FH	Accesses to the LLC, in which the data is present (hit) or not present (miss).
2EH	LLC Misses	41H	Accesses to the LLC in which the data is not present (miss).
C4H	Branch Instruction Retired	00H	Counts when the last uop of a branch instruction retires.
C5H	Branch Misses Retired	00H	Counts when the last uop of a branch instruction retires which corrected misprediction of the branch prediction hardware at execution time.

3) Event Register (Cont, Manual p.764)

- This table shows the value of Event Select bits and UMASK bits
- These values are already defined in the kernel, explanation is followed by the next slide

4. Source Code (Cont)

```
[EVENTS]
[PERF_COUNT_HW_CPU_CYCLES]                = 0x003c,
[PERF_COUNT_HW_INSTRUCTIONS]              = 0x00c0,
[PERF_COUNT_HW_CACHE_REFERENCES]          = 0x4f2e,
[PERF_COUNT_HW_CACHE_MISSES]              = 0x412e,
[PERF_COUNT_HW_BRANCH_INSTRUCTIONS]       = 0x00c4,
[PERF_COUNT_HW_BRANCH_MISSES]             = 0x00c5,
[PERF_COUNT_HW_BUS_CYCLES]                = 0x013c,
[PERF_COUNT_HW_REF_CPU_CYCLES]            = 0x0300,
/* pseudo-encoding */
```

Table 35-18. MSRs Supported by Intel® Processors
based on Intel® microarchitecture code name Sandy Bridge (Contd.)

186H	390	IA32_ PERFEVTSEL0	Thread	See Table 35-2.
187H	391	IA32_ PERFEVTSEL1	Thread	See Table 35-2.
188H	392	IA32_ PERFEVTSEL2	Thread	See Table 35-2.
189H	393	IA32_ PERFEVTSEL3	Thread	See Table 35-2.
18AH	394	IA32_ PERFEVTSEL4	Core	See Table 35-2; If CPUID.0AH:EAX[15:8] = 8
18BH	395	IA32_ PERFEVTSEL5	Core	See Table 35-2; If CPUID.0AH:EAX[15:8] = 8
18CH	396	IA32_ PERFEVTSEL6	Core	See Table 35-2; If CPUID.0AH:EAX[15:8] = 8

3) Event Register (Cont, Manual p.1438)

- These events are used and found in the Perf tool
- This is found in this directory: ~/Linux-4.19.120/arch/x86/event/intel/event/core.c
  * command [grep –r 'HW_CPU_CYCLES' ./*] in downloaded kernel folder (In my case, Linux-4.19.120 folder)
- the addresses for PERFEVTSEL registers are presented in the manual (image on the right)

4. Source Code (Cont)

```
132     /*****************************************
133      * IA32_PERFEVTSEL0: instructions
134      *****************************************/
135      msr = 0x186;
136      rdmsr(msr, mem_evtsel_low[0], mem_evtsel_high[0]);
137      printk("##READ BEFORE SETTING## IA32_PERFEVTSEL0 0x186: [low]%08x, [high]%08x\n", (mem_evtsel_low[0]), (mem_evtsel_high[0]));
138      low = 0x4300c0;
139      high = mem_evtsel_high[0] | 0x0;
140      wrmsr(msr, low, high);
141      printk("##WRITE## IA32_PERFEVTSEL0 0x186: [low]%08x, [high]%08x\n", (low), (high));
142
143      rdmsr(msr, low, high);
144      printk("##READ## IA32_PERFEVTSEL0 0x186: [low]%08x, [high]%08x\n", (low), (high));
```

3) Event Register (Cont)

- Checking PERFEVTSEL register by using system call rdmsr to the address 0x186
- Select an event from the previous slide and add the event value to the bit[23:16] (= 0x43), which enables EN, OS, and USR bits, resulting in 0x4300c0 in line 138 of the code
- Double check the value 0x4300c0 is written into the register by reading it again

4. Source Code (Cont)

Table 35-18. MSRs Supported by Intel® Processors
based on Intel® microarchitecture code name Sandy Bridge

C1H	193	IA32_PMC0	Thread	**Performance Counter Register** See Table 35-2.
C2H	194	IA32_PMC1	Thread	**Performance Counter Register** See Table 35-2.
C3H	195	IA32_PMC2	Thread	**Performance Counter Register** See Table 35-2.
C4H	196	IA32_PMC3	Thread	**Performance Counter Register** See Table 35-2.
C5H	197	IA32_PMC4	Core	**Performance Counter Register (if core not shared by threads)**
C6H	198	IA32_PMC5	Core	**Performance Counter Register (if core not shared by threads)**
C7H	199	IA32_PMC6	Core	**Performance Counter Register (if core not shared by threads)**
C8H	200	IA32_PMC7	Core	**Performance Counter Register (if core not shared by threads)**

```
253         /*********************************************
254          * IA32_PMCx reset to 0 to start counting
255          *********************************************/
256         msr = 0xC1;
257         low = 0x0;
258         high = 0x0;
259
260         for(i=0; i<EVTCOUNT; i++) {
261             wrmsr(msr, low, high);
262             ++msr;
263         }
```

4) PMCs (Manual p.1435)

- The address for PMCs starts from 0xC1
- Before reading these PMCs, reset the counters to 0
- After reset, place a program so that the PMCs count the occurrence of each event that is assigned in the previous step

4. Source Code (Cont)

Table 35-18. MSRs Supported by Intel® Processors
based on Intel® microarchitecture code name Sandy Bridge

C1H	193	IA32_PMC0	Thread	**Performance Counter Register** See Table 35-2.
C2H	194	IA32_PMC1	Thread	**Performance Counter Register** See Table 35-2.
C3H	195	IA32_PMC2	Thread	**Performance Counter Register** See Table 35-2.
C4H	196	IA32_PMC3	Thread	**Performance Counter Register** See Table 35-2.
C5H	197	IA32_PMC4	Core	**Performance Counter Register (if core not shared by threads)**
C6H	198	IA32_PMC5	Core	**Performance Counter Register (if core not shared by threads)**
C7H	199	IA32_PMC6	Core	**Performance Counter Register (if core not shared by threads)**
C8H	200	IA32_PMC7	Core	**Performance Counter Register (if core not shared by threads)**

```
253         /*********************************************
254          * IA32_PMCx reset to 0 to start counting
255          *********************************************/
256         msr = 0xC1;
257         low = 0x0;
258         high = 0x0;
259
260         for(i=0; i<EVTCOUNT; i++) {
261             wrmsr(msr, low, high);
262             ++msr;
263         }
```

4) PMCs (Manual p.1435)

- The address for PMCs starts from 0xC1
- Before reading these PMCs, reset the counters to 0
- After reset, place a program so that the PMCs count the occurrence of each event that is assigned in the previous step

4. Source Code (Cont)

```
277     low = 0, high = 0;
278
279     /*****************************************
280     * IA32_PERF_EVTSELx register
281     * BIT22: Enable Counters (Disabled)
282     *****************************************/
283     msr = 0x186; // EVTSEL register
284     for(i=0; i<EVTCOUNT; i++) {
285         wrmsr(msr, mem_evtsel_low[i], mem_evtsel_high[i]);
286         ++msr;
287     }
288
289     /*************************************************
290     * IA32_PERF_GLOBAL_CONTROL register: PMC disable
291     *************************************************/
292     msr = 0x38F;
293     wrmsr(msr, low, high);
294
295     /*****************************************
296     * IA32_PMCx reading
297     *****************************************/
298     msr = 0xc1;
299     for(i=0; i<EVTCOUNT; i++) {
300         rdmsr(msr, low, high);
301         printk("##READ## IA32_PMC%d 0x%02x: [low]%u, [high]%u\n", i, msr, low, high);
302         ++msr;
303     }
```

5) PMCs (Stop counting)

- By disabling either PERF_EVTSELx registers or PERF_GLOBAL_CTRL register, PMCs stop counting
- Mem_evtsel_low and mem_evtsel_high have the value before a user changes the value of the register; this prevents the kernel from panic by conserving the other bits except the bits we plan to change
- From slide 11, when printing in the kernel before changing the value, PERF_GLOBAL_CTRL register reports 0. This register manages enable/disable of other registers, so that writing 0 back seems to be safe
- Do not extract this part in a separate function because PMCs keeps counting until the EVTSELx and GLOBAL_CTRL registers are disabled

4. Source Code (Cont)

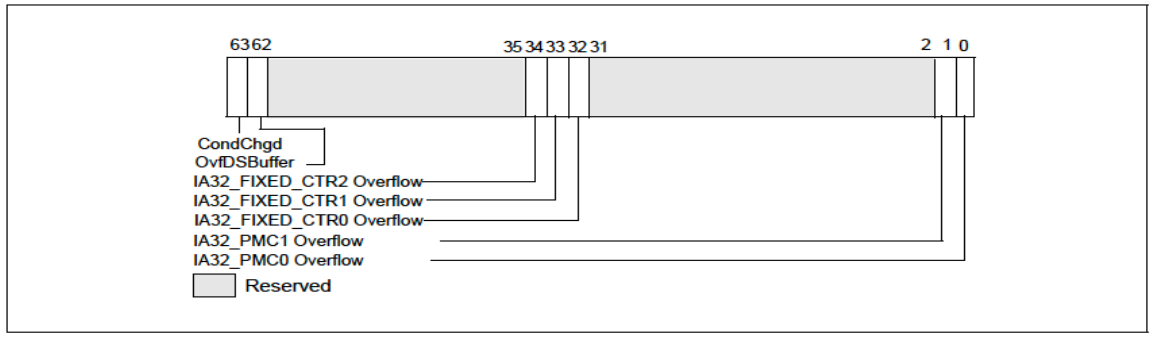

Figure 18-4. Layout of IA32_PERF_GLOBAL_STATUS MSR

Table 35-18. MSRs Supported by Intel® Processors based on Intel® microarchitecture code name Sandy Bridge (Contd.)

38EH	910	IA32_PERF_GLOBAL_ STATUS		See Table 35-2. See Section 18.4.2, "Global Counter Control Facilities."
		0	Thread	**Ovf_PMC0**
		1	Thread	**Ovf_PMC1**
		2	Thread	**Ovf_PMC2**

```
307     /*****************************************************
308      * IA32_PERF_GLOBAL_STATUS check PMCs overflow
309      *****************************************************/
310      msr = 0x38E;
311      rdmsr(msr, low, high);
312      printk("##CHECK OVERFLOW## IA32_PERF_GLOBAL_STATUS 0x38F: [low]%08x, [high]%08x\n", low, high);
```

6) Global Status Register (Left: Manual p.649, Right: Manual p.1444)

- In case PMCs overflow while counting, check the overflow bit

4. Source Code (Cont)

```
315     /*****************************************************
316      * IA32_PERF_EVTSELx check reset bits
317      *****************************************************/
318     msr = 0x186;
319     for(i=0; i<EVTCOUNT; i++) {
320         rdmsr(msr, low, high);
321         printk("@@CLEAR BITS@@ IA32_PERF_EVTSEL%d 0x%03x: [low]%08x, [high]%08x\n", i, msr, low, high);
322     }
323
324     /*****************************************************
325      * IA32_PERF_GLOBAL_CONTROL check reset bits
326      *****************************************************/
327      msr = 0x38F; // GLOBAL_CTRL register
328      rdmsr(msr, low, high);
329      printk("@@CLEAR BITS@@ IA32_PERF_GLOBAL_CTRL 0x38F: [low]%08x, [high]%08x\n", (low), (high));
330     return  0;
```

7) EVTSELx and GLOBAL_CTRL registers

- Check those two registers are cleared

5. Run The Program In Kernel

1) Before inserting the module, this step must be done:
   - Command ‘sudo modprobe msr’
   - This is also a module that allows to use wrmsr and rdmsr instructions
2) Type ‘make’: It will create an insertable module that has ‘.ko’ file format

   *Makefile has been attached
3) Type ‘sudo insmod filename.ko’ to insert the module
4) Type ‘sudo rmmod filename’ to remove the module from the kernel
5) Type ‘dmesg’ to check the print statements in the kernel

   *To observe the live debugging screen, type ‘dmesg –wH’