Validation of Performance Monitoring Architecture in Fullchip Environment

Major Project Report

Submitted in partial fulfillment of the requirements
for the degree of

Master of Technology
In
Electronics & Communication Engineering
(VLSI Design)
By

Prakruti R Desai
(17MECV04)

Electronics & Communication Engineering Department
Institute of Technology
Nirma University
Ahmedabad - 382 481
May, 2019

Validation of Performance Monitoring Architecture in Fullchip Environment

Major Project Report

Submitted in partial fulfillment of the requirements
for the degree of

Master of Technology
In
Electronics & Communication Engineering
(VLSI Design)
By

Prakruti R Desai
(17MECV04)

Under the Guidance of

Internal Guide
Dr Amisha Naik
Associate Professor (VLSI Design)
Nirma University

External Guide
Mr. Mahesh V Gaidhani
Engineering Manager
Intel Technology India
Pvt Ltd.

Electronics & Communication Engineering Department
Institute of Technology, Nirma University
Ahmedabad - 382 481
May, 2019

Declaration

This is to certify that

1. The thesis comprises my original work towards the degree of Master of Technology in VLSI Design at Nirma University and has not been submitted elsewhere for a degree.

2. Due acknowledgment has been made in the text to all other material used.

Prakruti R Desai
(17MECV04)

NIRMA UNIVERSITY

INSTITUTE OF TECHNOLOGY

NAAC ACCREDITED ‘A’ GRADE

Certificate

This is to certify that the Major Project entitled **“Validation of Performance Monitoring Architecture in Fullchip Environment”** submitted by **Prakruti R Desai (17MECV04)**, towards the partial fulfillment of the requirements for the degree of Master of Technology in VLSI Design, Nirma University, Ahmedabad is the record of work carried out by her under our supervision and guidance. In our opinion, the submitted work has reached a level required for being accepted for examination.The results embodied in this major project, to the best of our knowledge, haven’t been submitted to any other university or institution for award of any degree or diploma.

Dr Amisha Naik	Dr N. M. Devashrayee
Internal Guide	PG Coordinator (VLSI Design)
Dr D. K. Kothari	Dr Alka Mahajan
Head, EC Dept.	Director, IT - NU
Date :	Place : Ahmedabad

Acknowledgment

Let me take the opportunity to express my deep regards to Mr. Mahesh. V. Gaidhani (Project Manager) for assigning me such project and providing his guidance and constant encouragement during the project. I would also like to thank Sudarshan Nayak (Mentor) for his guidance, help and inspiring me to put my best efforts.

I would like to express my gratitude & sincere thanks for generous assistance to my guide Dr. Amisha Naik, Associate Professor, VLSI Design, Institute of Technology, Nirma University, Ahmedabad for his guidance and constant encouragement during my course of project. Special thanks to all faculties who has always been an inspiration and guided us with their experience.

I would like to express my gratitude towards my parents for constant support and encouragement in life. I also wish to express my heartfelt appreciation to my friends and colleagues at Intel who have rendered their support throughout my project, both explicitly and implicitly.

- Prakruti R Desai

(17MECV04)

Abstract

With increasing number of cores in multi core processor system, focusing on its functional verification is not enough because the motivation for building such systems is to achieve high levels of system throughput. A functionally correct SoC with poor performance will fail in market. So performance monitoring of such system is an important task before its tape out. Furthermore, focusing on individual system components for performance monitoring is not sufficient. In fact, UNCORE performance is bottleneck for entire system performance. So focusing on system performance is significant.

With advance in VLSI Technology these days, high density processors enter into market which provides luxury of hardware Performance Monitoring Unit inside the chip that monitors system performance accurately. So validating this Performance monitoring unit is overriding task to get reliable information of device performance.

This thesis outlines the architecture of Performance monitoring unit (PMU) in SoC and functional validation of PMU. It covers interrupt based verification and includes flow to validate core and uncore PMU, core and uncore events responsible for performance, validating Performance monitoring counter and registers, coverage implementation and improving coverage at fullchip level.

Table of Contents

Abstract i

Table of Contents iv

List of Figures v

List of Abbreviation vi

1 Introduction **1**

1.1 Motivation 2

1.2 Objective 3

1.3 Overview of the Thesis 4

2 Literature Survey **5**

2.1 SoC Architecture 5

2.1.1 SoC block Diagram 6

2.1.2 Flow of data transaction in SoC: 8

2.2 Performance monitoring Overview 8

2.2.1 Architectural Performance monitoring 9

2.2.2 Architectural PMON vesion 1 10

2.2.3 Bit field layout of PM contol register $IA32\_PERFEVESETxMSRs$ 11

2.2.4 Pre defined architectural performance events 14

2.2.5 Types of Performance Measurement 15

3 Performance Monitoring of INTEL Processor 18

3.1 Performance Monitoring of Processor Core 19

3.1.1 Processor Event Based Sampling (PEBS) 20

4 PMON Validation strategy 21

4.1 PMON validation environment 21

4.2 PMON validation methods 22

4.3 Validation Scope 23

4.4 Validation flow of Performance Monitoring Unit 23

4.5 PMON validation test flow 25

5 Result and Discussion 27

5.1 logbook results 27

5.2 PMON test Results 29

5.3 Error Generation and Challanges 30

5.4 Conclusion 31

5.5 Future Scope 31

References 32

List of Figures

1.1.1 Memory wall 3

2.1.1 SoC architecture 6

2.2.1 layout of $IA32\_PERFEVESETxMSRs$ 12

3.1.1 Bit field layout of $IA32\_PEBS\_ENABLE$ 20

4.4.1 validation of uncore PMON 24

4.5.1 uncore PMON validation test flow 25

5.1.1 Logbook Summary of Passing log 27

5.1.2 log report 28

5.2.1 Counter register read value 29

5.2.2 Configuring event select Registers for upcoming event 29

5.3.1 Logbook summary of failing Log 30

5.3.2 PMON failed as handle was not called on PMI generation 30

List of Abbreviation

PMU	- Performance Monitoring Unit
MSR	- Model Specific Register
PMI	- Performance Monitoring Interrupt
uC	- Microcontroller
uP	- Microprocessor
SoC	- System on Chip
PSoC	- Programmable System on Chip
FPGA	- Field Programmable Gate Array
PMON	- Performance monitoring
PMC	- Performance monitoring counter
PEBS	- Processor Event Based Sampling
DUT	- Device Under Test
PMON	- Performance Monitoring
PMU	- Performance Monitoring Unit
CPU	- Central Processing Unit
USB	- Universal Serial Bus
JTAG	- Joint Test Action Group

Chapter 1

Introduction

Complexity in the current generation of multicore systems-on-chip is staggering: multiple processors, multiple levels of on-chip cache, high-bandwidth interconnect and memory interfaces, along with sophisticated application-specific accelerators and various input/output interfaces. Driving this complexity is the need for ever-increasing system throughput. Prohibitive mask set costs and time to market pressures further motivate the need to get it right the first time". Therefore, SoC performance verification before final tape out is more critical than ever. Pre-silicon performance verification focuses on ensuring that the SoC meets performance criteria for complex, real-world applications, in contrast to verifying that models of the SoC accurately predict performance, or that subsystems of the design meet latency or timing requirements, or that the manufactured parts meet performance criteria. Furthermore, formal techniques are not yet ready for efficient use by industry. For these reasons we concentrate on SoC performance verification in fillchip environment.

Earlier, processors do not have luxuries of hardware performance monitoring unit on the system. During those days to monitor performance of the system software simulator was used. Workload was uploaded on simulator and the result of simulation predicts the overall performance of system. System performance include performance of cpu, memory, storage disk. But monitoring the performance through software simulation does not give accurate result.

With advance in technology and increased density in processors,hardware performance monitoring unit or the PMU is found in all high end processors these days. The performance related information stored in hardware registers is read by software. So the system performance information obtained is more accurate. Performance monitoring unit is a concept to monitors the PMON events for better performance of system. We can measure parameters like instruction cycles, cache hits, cache misses, branch misses and many others depending on the support.

1.1 Motivation

Over years the enhancement in performance has improved fast but memory technology has not improved at that high rate. This gap between processor and memory performance is called memory wall. The motivation is to bridge the gap of memory wall. And to have high performance and fast computer processors in this digital revolution era.

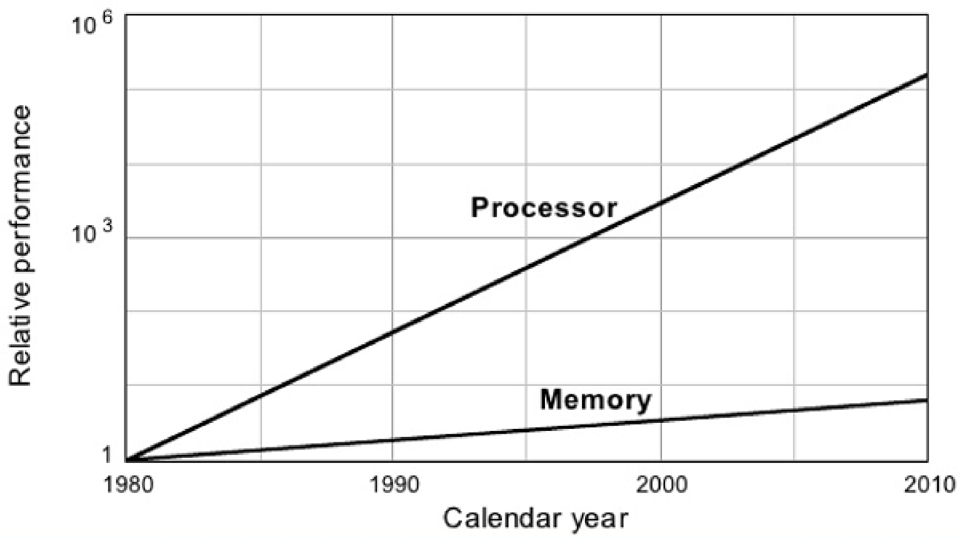


Figure 1.1.1: Memory wall

1.2 Objective

The main objective is to understand Performance monitoring unit at architectural level and flow of validating it.

- To understand the test environment at fullchip level and SoC level.
- To understand architecture of performance monitoring unit of core and uncore.
- Validating the architectural performance monitoring events in fullchip environment
- Validating capacity of core to handle Performance Monitoring Interrupt and perform subroutine.

1.3 Overview of the Thesis

Chapter wise overview of this thesis is shown below:

Chapter 2 Covers the Literature Survey which will discuss about SoC architecture and PMU overview.

Chapter 3 Covers detail of performance monitoring monitoring unit of INTEL processor using microarchitecture named Nehalem. This includes different methods and approaches for performance monitoring in core and uncore part of SoC.

chapter 4 Covers validation strategy for validating performance monitoring unit

chapter 5 Covers results and discussion

Chapter 2

Literature Survey

2.1 SoC Architecture

A system on a chip is an integrated circuit that integrates all components of a computer or other electronic system on a single substrate. These components typically include a central processing unit (CPU), memory, display and graphics, USB, JTAG and other peripherals. It may contain digital, analog, mixed-signal, and often radio frequency signal processing functions, depending on the application. As they are integrated on a single substrate, SoCs consume much less power and take up much less area with quite good performance than multi-chip designs with equivalent functionality. Because of this, SoCs are very common in the mobile computing and edge computing markets.Systems on chip are commonly used in embedded systems and the Internet of Things.

In general, there are four different types of SoCs:

- SoCs that are built around a microcontroller (C),

- SoCs that are built around a microprocessor (P), often found in mobile phones
- Specialized SoCs designed for specific applications that do not fit into the above two categories, and
- Programmable systems-on-chip (PSoC), where most functionality is fixed but some functionality is reprogrammable in a manner analogous to a field-programmable gate array(FPGA).

2.1.1 SoC block Diagram

below figure shows blocks diagram of SoC

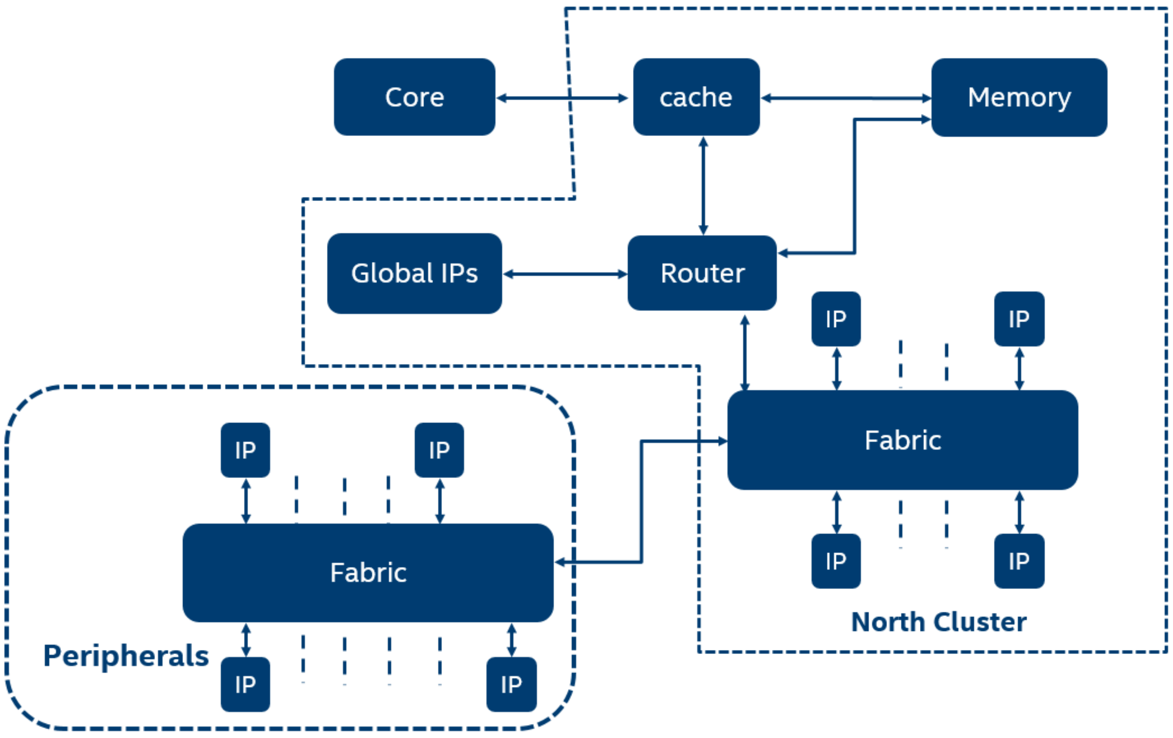


Figure 2.1.1: SoC architecture

SoC is divided into two prts, Core and UNCORE. Core is the main processing part that includes processing and graphic core. Apart from core, all other modules are in uncore part of SoC. This include base die and compute die. Compute die also called as north cluster includes global IP's, cache, memory, and other local IP's. Base die includes peripheral IP's.

- **Core** is main processing unit of sytem. There may be single core or multicore depending upon system.
- **Cache :**It is a small sized memory with high speed placed between core and main memory. Frequently used data is stored in cache to increase the speed. There are three levels of cache L1, L2 and L3. Where, L1 and L2 are dedicated to each core in multicore system and L3 is shared among all cores.
- **Memory :**It is the main memory unit that stores the data and instructions. Data that is to be processed is read from this memory and processed data is written in it. It is larger sized memory.
- **Fabric :**It is a type of router with muxing logic that routs the data coming from the IPs connected to it depending upon the priority.
- **Router :**It is the main routing unit of entire system. Data transaction from any of the subsystem reaching to the core is routed by router.
- **Peripherals :**This includes all peripheral IPs. Also called as south cluster or base die.
- **Interfaces :**These are connecting link between subsystems. There uses different protocols for transaction among different subsystems.

- **Global IPs :**These are the set of IPs that communicates with each of the subsystems like power, clock, security, fuse, etc.

2.1.2 Flow of data transaction in SoC:

For read and write of data: Data that is required for processing is requested to router. This data is snooped in cache, if it is found then that event is cache hit and data is directly fetched from cache. This saves clock cycles to fetch the data from memory. As a result operation performed is faster. If the data is not found in cache then it will be cache miss and sends the request to fetch data from memory.

Data transaction from other IPs: If the data from peripheral IP is transmitted to the core, then data transmission will be as follows. Data from the peripheral IP transmits forward to fabric1 depending upon the priority given by fabric0. This fabric decides that among the data from other IPs which one is given priority and according to their priority order data is transmitted. Fabric1 sets the priority order of the data from fabric0 and other IPs associated with it. This data is transmitted to router. router will route based on physical address of the data whether it is sent to memory or core. If the request is sent from IP then which data is to be transmitted either from memory or core or cache or global IP. And the fetched data is sent via same route.

2.2 Performance monitoring Overview

The complexity of computing systems has tremendously increased over the last decades. Hierarchical cache subsystems, non-uniform memory, simultaneous multithreading and out-of-order execution have a huge impact on the performance and compute capacity of modern processors.

Pmon consists a set of model specific pmon counters MSRs that counts performance monitoring events that are to be monitored and measured.Performance monitoring capabilities are of two classes: first class uses non-Architectural performance monitoring events. These are model specific and vary with enhancement. This uses counting and interrupt based event sampling. These cannot be enumerated using CPUID. Second class is architectural performance monitoring events. This also support counting and interrupt based event sampling. The behavior of this events is consistent across processor implementation. Availability of this events is enumerated with CPUID 0AH.

2.2.1 Architectural Performance monitoring

In this class of performance monitoring monitoring, Events are consistent throughout processor microarchitecture. CPUID.0AH provides version ID. Version with ID 1 is the base version and advanced one has ID 2,3,4.

- INTEL core solo and core duo processor support functionality with version ID 1 which is called base level functionality.
- Core 2 duo processor with T7700 series and newer processor based on core microarchitecture support functionality of Version ID 1 and 2.
- 45nm atom processor,32nm atom processor based on silvermont microarchitecture and atom processor based on airmont microarchitecture support functionality of Version ID 1,2,3.

- CPUID.0AD:EAX[7:0] reports Version ID=3
- Core processor and related Xenon processor families based on nehalem through broadwell uarchi support Version ID 1,2,3
- Processor based on skylake and kabylake micro architecture support VID 4.

2.2.2 Architectural PMON vesion 1

Configuring pmon architectural events means programming pmon event select registers. There are few set of PMON event select reg MSRs ($IA32\_PERFEVESETxMSRs$). Result of pmon event select register configuration is reported to PMC ($IA32\_PMCxMSR$), which is performance monitoring counter. PMC is paired with PMON event select register.

Below points justifies how PMON counters and PMON event select register are architectural:

- Bit field layout of all PMON event select register is consistent throughout micro architecture.
- Address of all the PMON event select register remains same throughout microarchitecture.
- Address of all PMON counter remains same throughout microarchitecture.
- Each logical processor sharing same physical core has their own set of PMON counters and PMON event select reg. Logical processors sharing same core do not share counter and configuration facilities.

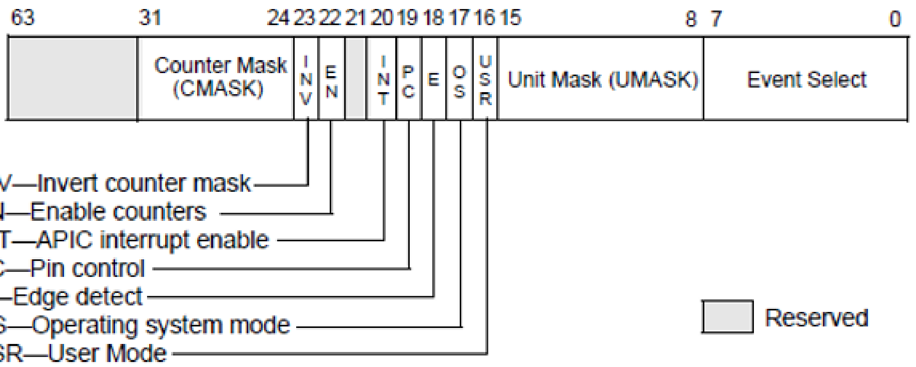
For counting the following information, architectural performance monitoring provides CPUID mechanism:

- No. of PMON counters available in logical processor.
- No. of bits supported in each counter.
- No of architectural PMON events supported in logical processor.

Software uses CPUID.0AH to find out availability of PMON architectural event. Corresponding to the version number of architectural performance monitoring an identifier is provided by PMON leaf.Software ask CPUID.0AH for version identifier. If version identifier is greater than zero, then architectural PMON capability is supported. It then analyses CPUID.0AH.EAX, CPUID.0AH.EBX for facilities available. In the initial implementation of architectural PMON, it is determined by the software that how many counter and register pairs are accommodated by the core, number of bit width of counter and number of architectural events occurring.

2.2.3 Bit field layout of PM contol register $IA32\_PERFEVESETxMSRs$

Set of PMON counter and corresponding PMON event select register are included in facilities of architectural PMON. Their properties are as follows: PMC starts at address 0C1H, and using CPUID.0AH.EAX[15:8] number of logical processor is obtained. Event select register starts with address 186H and each register is paired with corresponding PMC in 0C1H address block. CPUID.0AH.EAX[23:16] reports the bit width of PMC



archi.PNG

Figure 2.2.1: layout of $IA32\_PERFEVESETxMSRs$

- **Event Select Fields (bits 0-7) :** This field is used to select the logic unit to detect the performance monitoring event to be monitored. So the values to be filled in this field is determined by the architecture.

- **Unit mask (UMASK) fields (bits 8-15):** The logic unit selected by the Event Select field might be capable of monitoring multiple events. So this UMASK field is used to select one of those events which can be monitored by the logic unit. So based on the logic unit selected the UMASK field may have one fixed value or multiple values, which is dependent on the architecture.

- **USR flag (bit 16):** This flag, if set, tells the logic unit to monitor events which happen when the processor is running in the User privilege level i.e. levels 1 through 3.

- **OS Flag (bit 17):** This flag, if set, tells the logic unit to monitor events which happen when the processor is running in the highest privilege level i.e. level 0. This flag and the USR flag can be used together to monitor or count all the events.

- **E (Edge Detect) (bit 18):** This flag when set counts the number of times the selected event has gone from low to high state.

- **PC (Pin Control) (bit 19):** This flag when set increments the counter and toggles the PMi pin when the monitored event occurs. And when not set, it toggles the PMi pin only when the counter overflows.

- **INT (APIC interrupt enable) flag (bit 20):** When this flag is set the processor raises an interrupt when the performance monitoring counter overflows.

- **EN (Enable Counters) flag (bit 22):** This flag when set enables the performance monitoring counters for the event and when clear disables the counters.

- **INV (inversion) flag (bit 23):** This flag when set inverts the output of CMASK comparison. This enables the user to set both greater than and less than comparisons between CMASK and the counter value.

- **CMASK (Counter mask) field (bits 24 through 31):** If this field has a value more than zero then that value is compared to the number of events generated in one clock cycle. If the events generated is more than the CMASK value then the counter is incremented by one else the counter is not incremented.

Other features provided by hardware:

- **Fixed function performance counter register and associated control register:** There are a few counters which can measure only a specific event unlike general purpose counters which can be configured to measure different events.

- **Global Control Registers:** Some architectures provide global control registers which can be used to control all or a group of control registers or counters. This reduces the number of instructions required to modify the control registers and hence eases programming.

2.2.4 Pre defined architectural performance events

A processor that supports architectural performance monitoring may not support all the predefined architectural performance events. The behavior of each architectural performance event is expected to be consistent on all processors that support that event. Minor variations between microarchitectures are noted below

- **UnHalted Core Cycles** Event select 3CH, Umask 00H.This event counts core clock cycles when the clock signal on a specific core is running (not halted). The performance counter for this event counts across performance state transitions using different core clock frequencies.

- **Instructions Retired** Event select C0H, Umask 00H. This event counts the number of instructions at retirement. For instructions that consist of multiple micro-ops,this event counts the retirement of the last micro-op of the instruction.

- **UnHalted Reference Cycles** Event select 3CH, Umask 01H. This event counts reference clock cycles at a fixed frequency while the clock signal on the core is running. The event counts at a fixed frequency, irrespective of core frequency changes due to performance state. transitions.

- **Last Level Cache References** Event select 2EH, Umask 4FH. This event counts requests originating from the core that reference a cache line in the last level cache.

- **Last Level Cache Misses** Event select 2EH, Umask 41H. This event counts each cache miss condition for references to the last level cache.
- **Branch Instructions Retired** Event select C4H, Umask 00H. This event counts branch instructions at retirement. It counts the retirement of the last micro-op of a branch.
- **All Branch Mispredict Retired** Event select C5H, Umask 00H. This event counts mispredicted branch instructions at retirement.

2.2.5 Types of Performance Measurement

Counting

In this type of measurement the total number of events that happen in a given time duration are aggregated and reported at the end of the duration. The performance control registers are set for counting the desired event and after the end of monitoring period the values of these registers are read.

Challenges faced:

The challenges faced uring counting are 1) the number of events to be monitored are more than the total number of counters provided by the processor and 2) Different events to be monitored are measured by the same digital logic present in the processor.

Resolution:

Multiplexing is used to resolve the issues mentioned above. In the first case, as the number of counters is less the events share time of the same counter i.e. time division multiplexing. Which means that one event does not get a dedicated counter for the entire duration of measurement. Instead the events are measured in small durations many times during the entire measuring period. At the end of measurement duration the actual measurement period is also recorded and the aggregated event count is scaled for the complete measurement period. In the second case, the same technique is used as in the first case. The only difference being that this time the digital logic unit is time multiplexed to measure different events.

Limitations:

Although multiplexing solves some issues and will be pretty close to the actual values but the scaled results are not completely reliable. It may so happen that the event which was not being measured for a particular instance may have spiked or tanked during that instance and the scaled value will be misleading. So, in cases where highly precise values are desired the user should take care that the monitored event gets dedicated hardware and is not time multiplexed.

Event based Sampling

In this type of measurement, the PMU counters are configured to overflow after a preset number of events and when the overflow happens the process status information is recorded by capturing the data of the instruction pointer, general purpose registers and EFLAG registers. This sampled data can be utilized for profiling software applications, finding how the software is utilizing the underlying hardware and many other purposes.

Limitations

- **Sampling Delay:**There is delay between the counter overflow and the time when interrupt is raised. This combined with the long pipelines present in high-end processors the program counter data stored at the time of sampling may not be the event which caused the counter to overflow.

- **Speculative count**Most high end processors these days use branch predictions which leads to speculative execution of instructions which may not complete if some other branch is selected. But these speculatively executed instructions may cause events and contribute to the event count even if they do not complete, which is not correct.

Chapter 3

Performance Monitoring of INTEL Processor

In this chapter Performance Monitoring of processors based on Intel's Nehalem microarchitecture is described in detail.

PMU uses general purpose and fixed function performance monitoring counters in the processor core. Along with this counters, PMU uses MSR registers to configure and count the events occurring in core and uncore. Based on occurrence of events, these events are categorized into two broad categories.

- **Core events:** This include all the events occurring in core of a multicore processor and the set of events interfacing multiple cores and uncore from core side.
- **Uncore events:** these events are the events that occur outside core and specific to uncore and monitored by uncore PMU. Uncore also called off-core is shared by multiple cores in multi-core processor.

3.1 Performance Monitoring of Processor Core

To enhance the performance monitoring of events related to processor core it includes the following facilities.

There are four general purpose counters $IA32\_PMCx$, MSR registers associated with counter, $IA32\_PERFEVESETxMSRs$ to configure the events to be monitored, global counters and MSR registers associated with it.The width of the hardware supported counter has been increased to 48 bits. For all the types of events architectural or non-architectural, each of these counters used in PMU supports PEBS (Processor events based sampling) facilites and thread qualification. PEBS in the performance monitoring unit of this microarchitecture has been enhanced to include new format like load latency.

It also includes the facility of load latency sampling. The memory load operation latency can be sampled using this facility. It includes measure of load latency of complete memory transaction initiated and return back to memory subsyatem.

It also includes countng of core dependent uncore events. Certain events are communicating between core and uncore. This performance monitoring unit has facility to monitor such events. For this purpose two additional configuration registers $IA32\_PERFEVESETxMSRs$ are also assigned to select and configure these events.

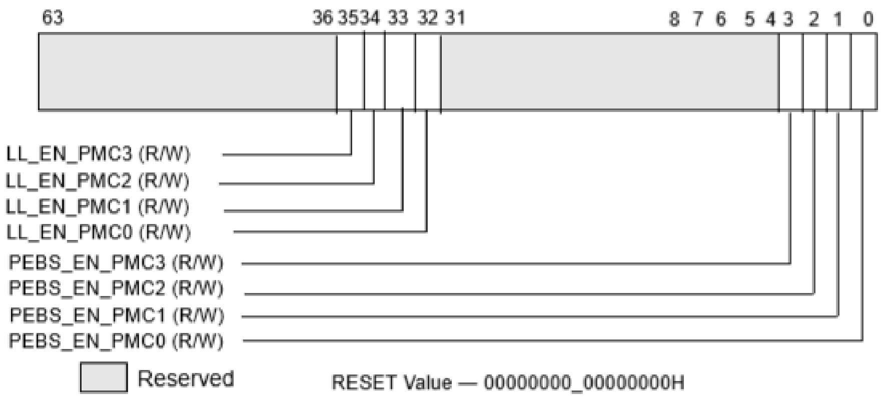


Figure 3.1.1: Bit field layout of $IA32\_PEBS\_ENABLE$

3.1.1 Processor Event Based Sampling (PEBS)

PEBS is used by all general purpose counters $IA32\_PMCx$ if the events that are to be counted supports PEBS. To detect whether performance monitoring and PEBS is supported by the processors, software uses two bits $IA32\_MISC\_ENABLE[7]$ and $IA32\_MISC\_ENABLE[12]$. It uses register $IA32\_PEBS\_ENABLE$ to configure PEBS. Additionaly, latency record can also be captured from PEBS records. As shown in figure 3.1.1, four bits of register $IA32\_PEBS\_ENABLE$ are dedicated to $IA32\_PMCx$ overflow condition and other four bits are dedicated for latency record.

Chapter 4

PMON Validation strategy

This chapter includes validation of Performance Monitoring Unit in different environment, different methods of validation and test case approach for better and bug free functional verification.

4.1 PMON validation environment

Performance monitoring unit can be validated in fullchip environment and SoC level environment.

- **SoC level environment** In SoC level environment, actual core model is not used to validate the design. Instead, BFM (Bus Functional Model) of core model is used.simulation method is preferred here. As this approach saves cost and simulation time since actual model is not used majority of the events monitoring is done through this approach. But if there is interaction between core and uncore it needs to be verified at fullchip level. So customer visible events are monitored in this environment.

- **Fullchip environment** In fullchip level actual core model is used to validate the design. For certain events interacting between core and uncore, this approach is followed. It uses emulation method and test are run on fpga board. this gives accurate result in less time and entire design is hardware verified. But this approach is too costly. So all events are not monitored at fullchip level.

4.2 PMON validation methods

validation of PMON is done through simulation and emulation

To validate functional behaviour of a design, we go for software simulation approach. If the design size is manageable and it takes fair amount of simulation run time, then this is best suited option as its cost is less compared to emulation. Simulation is easy to setup and use. But to verify larger designs, simulation time is very large. So we go for emulation method to reduce run time.

Hardware simulation of the design is called emulation.The piece of code is dumped on fpga board so the board act as DUT. This DUT is verified. When design size is large and accommodate large number of gates, emulation will be a faster approach as its run time is less. But its cost is too high. So simulation is also used with emulation to verify the design.

4.3 Validation Scope

validation of performance monitoring unit is done at core and uncore side.

- At core level, Performance monitoring is done using PEBS (Performance Event Based Sampling) and load latency performance monitoring.
- At uncore level performance monitoring is done by set of PM counters and event select performance monitoring MSR. It also uses interrupt based aproach for counter overflow.

4.4 Validation flow of Performance Monitoring Unit

For performance monitoring in uncore, performance monitoring counters and event select registers are located in concerned subsystems to monitor the events related to performance monitoring. The events to be monitored is configured by event select register. counter counts the configured event. As soon as the counter overflows, as shown in the figure it hits the PMI request to NCU and request for interrupt handle. NCU broadcast this request message to all the cores. The core having APIC controller will accept the request and provide interrupt handle for subroutine task.As interrupt is generated, counter resets to zero.

Interrupt generated is checked by self check mechanism. In this, virtual memory is created and when interrupt is generated some value is written to this memory. scanning this memory will verify generation of interrupt on counter overflow. If memory contains non zero value, interrupt is generated and if memory contains zero value interrupt is not

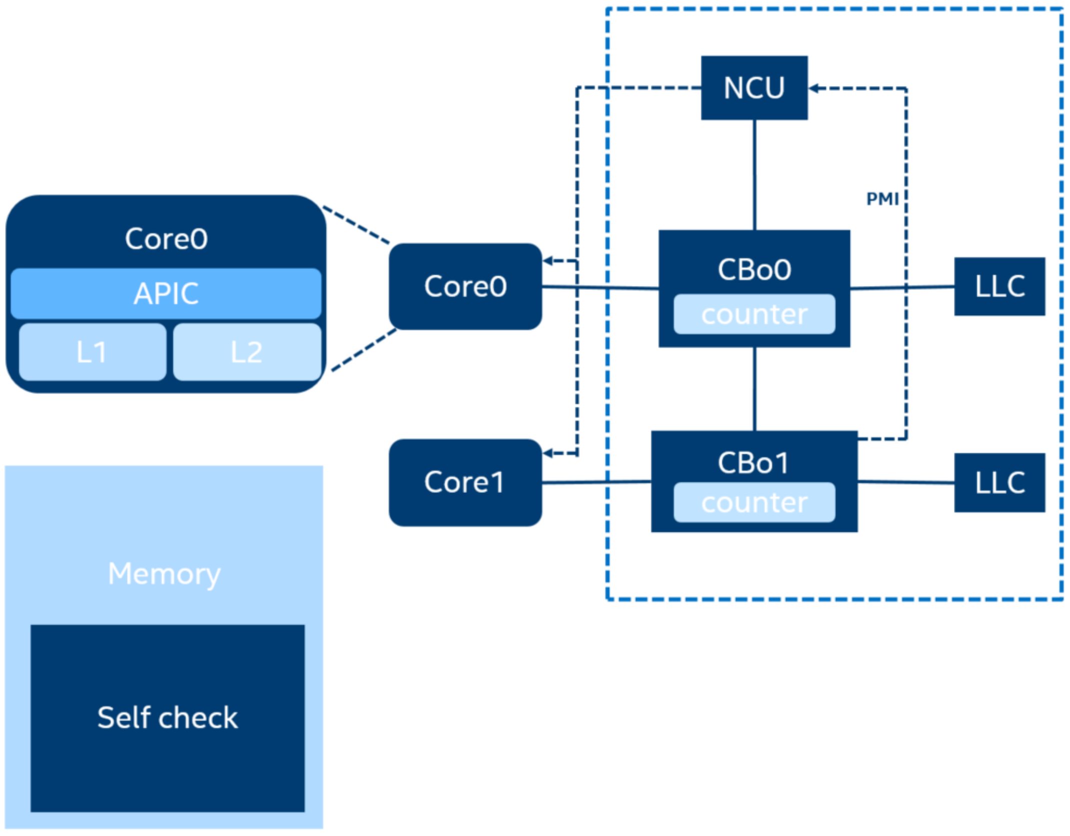


Figure 4.4.1: validation of uncore PMON

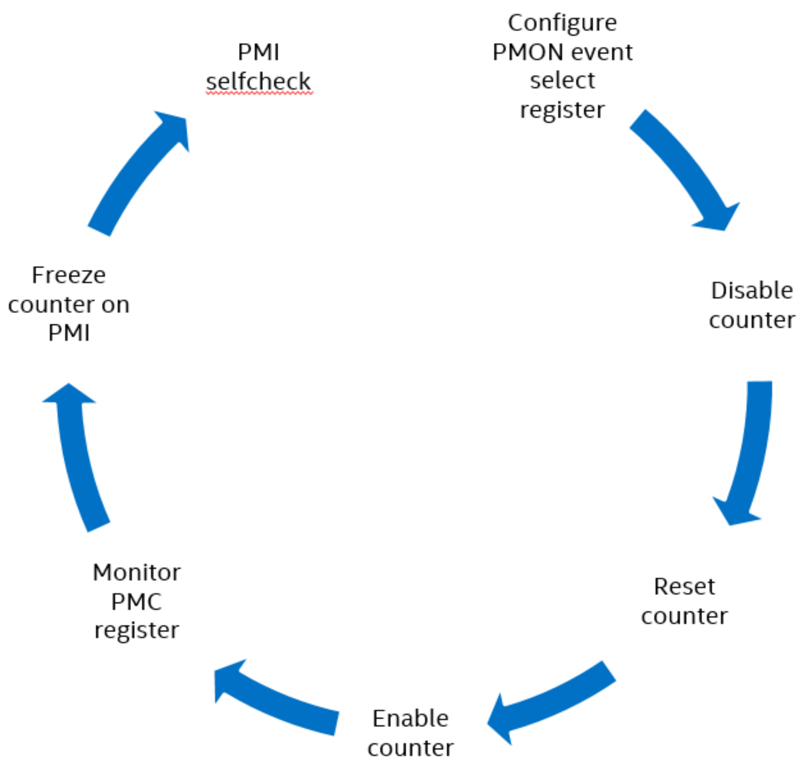


Figure 4.5.1: uncore PMON validation test flow

generated.

things to be verified

- Here we need to verify whether interrupt is generated when counter overflows.
- Correct interrupt request message is broadcast to all the cores by NCU.
- self-check mechanism works correctly.

4.5 PMON validation test flow

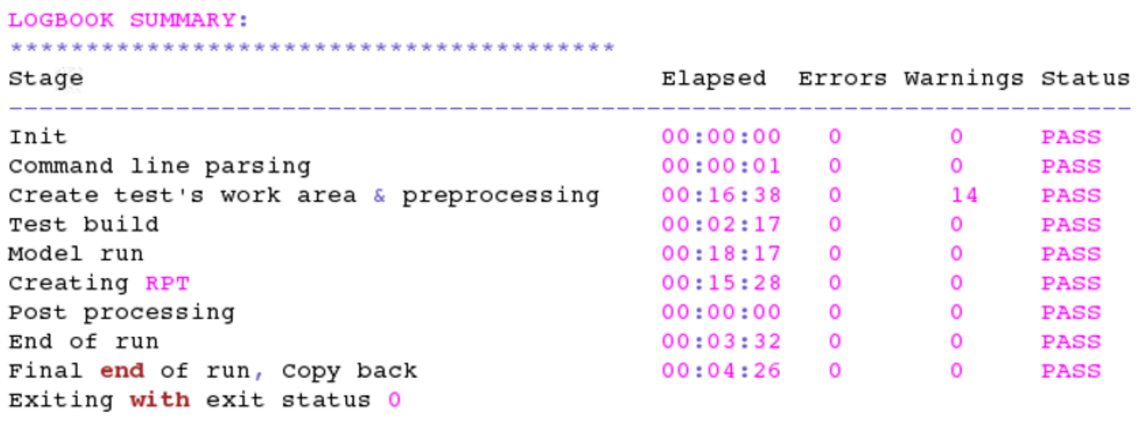
To verify PMU the first step is to configure performance monitoring event select registers that are located in subsystem for the events to be monitored. After register configuring, all the counters are globally disable and are reset to zero. To start monitoring the PMON events these counters are enabled. As soon as the counters are enabled it starts counting the occuring events and this is monitored by tracing the counter registers associated with the counters. As soon as this counter overflows it generates PMI interrupt request and freeze the counter to stop from further counting. This interrupt generation is verified by self check mechanism mentioned above. This is how uncore PMON is verified.

Chapter 5

Result and Discussion

5.1 logbook results

The test that is run passes through the following stages

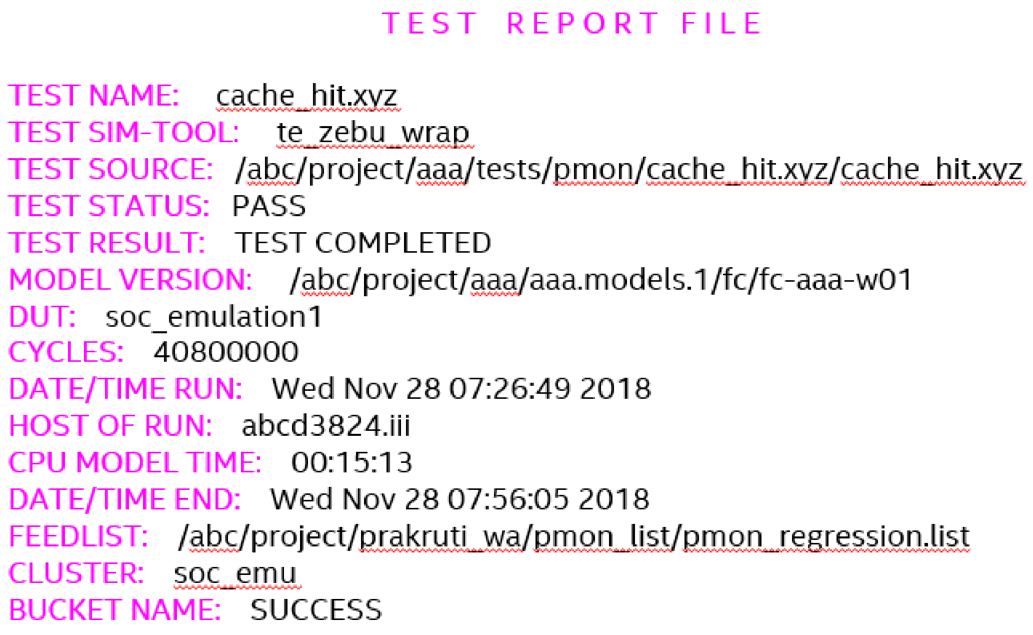


```
LOGBOOK SUMMARY:
*****************************************
Stage                                      Elapsed  Errors Warnings Status
---------------------------------------------------------------------------
Init                                       00:00:00   0       0      PASS
Command line parsing                       00:00:01   0       0      PASS
Create test's work area & preprocessing    00:16:38   0       14     PASS
Test build                                 00:02:17   0       0      PASS
Model run                                  00:18:17   0       0      PASS
Creating RPT                               00:15:28   0       0      PASS
Post processing                            00:00:00   0       0      PASS
End of run                                 00:03:32   0       0      PASS
Final end of run, Copy back                00:04:26   0       0      PASS
Exiting with exit status 0
```

Figure 5.1.1: Logbook Summary of Passing log

- **Command line Parsing:** In this stage we set the fuses using switches, doing that we disable or enable some features and depend upon test plan we can control how much part of test we want to run.
- **Create tests work area:** In this stage it set environment depend upon project.

- **Test build:** This stage compiler compile test case and convert higher level language to machine level language.
- **Model run:** At this stage converted machine level language load in emulation board and run test case on emulation board.
- **Creating RPT:** In this stage it create all transaction data trackers file which more use full for debug purpose, for coverage and checker.
- **Post processing:** At this stage all post process script (like coverage and checker script) are run.



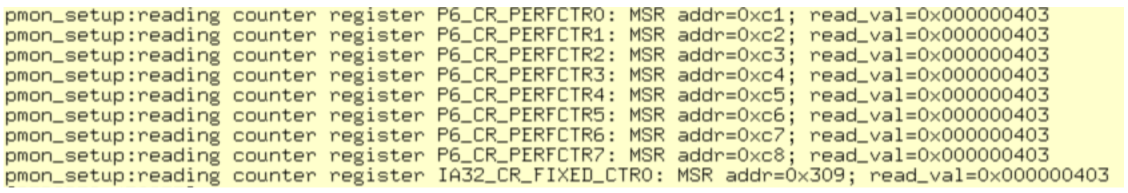
```
                    TEST  REPORT  FILE
TEST NAME:    cache_hit.xyz
TEST SIM-TOOL:    te_zebu_wrap
TEST SOURCE:  /abc/project/aaa/tests/pmon/cache_hit.xyz/cache_hit.xyz
TEST STATUS:  PASS
TEST RESULT:   TEST COMPLETED
MODEL VERSION:    /abc/project/aaa/aaa.models.1/fc/fc-aaa-w01
DUT:   soc_emulation1
CYCLES:   40800000
DATE/TIME RUN:   Wed Nov 28 07:26:49 2018
HOST OF RUN:   abcd3824.iii
CPU MODEL TIME:   00:15:13
DATE/TIME END:   Wed Nov 28 07:56:05 2018
FEEDLIST:   /abc/project/prakruti_wa/pmon_list/pmon_regression.list
CLUSTER:   soc_emu
BUCKET NAME:   SUCCESS
```

Figure 5.1.2: log report

The above figure shows test report from logbook. The data shown in the result are reference numbers and not real.

5.2 PMON test Results

Result Analysis 1

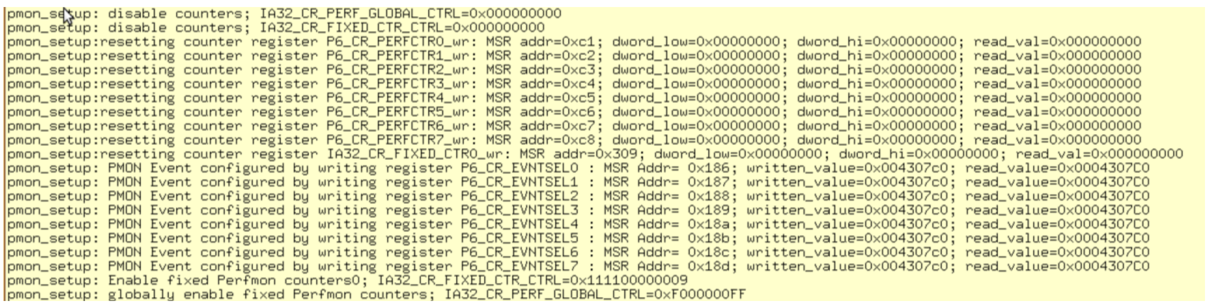


```
pmon_setup:reading counter register P6_CR_PERFCTR0: MSR addr=0xc1; read_val=0x000000403
pmon_setup:reading counter register P6_CR_PERFCTR1: MSR addr=0xc2; read_val=0x000000403
pmon_setup:reading counter register P6_CR_PERFCTR2: MSR addr=0xc3; read_val=0x000000403
pmon_setup:reading counter register P6_CR_PERFCTR3: MSR addr=0xc4; read_val=0x000000403
pmon_setup:reading counter register P6_CR_PERFCTR4: MSR addr=0xc5; read_val=0x000000403
pmon_setup:reading counter register P6_CR_PERFCTR5: MSR addr=0xc6; read_val=0x000000403
pmon_setup:reading counter register P6_CR_PERFCTR6: MSR addr=0xc7; read_val=0x000000403
pmon_setup:reading counter register P6_CR_PERFCTR7: MSR addr=0xc8; read_val=0x000000403
pmon_setup:reading counter register IA32_CR_FIXED_CTR0: MSR addr=0x309; read_val=0x000000403
```

Figure 5.2.1: Counter register read value

As shown in figure 5.2.1, the number of Performance monitoring events that are counted by the performance monitoring counter (general and fixed counters) is stored in counter register. This figure shows reading value of counter register.

Result Analysis 2



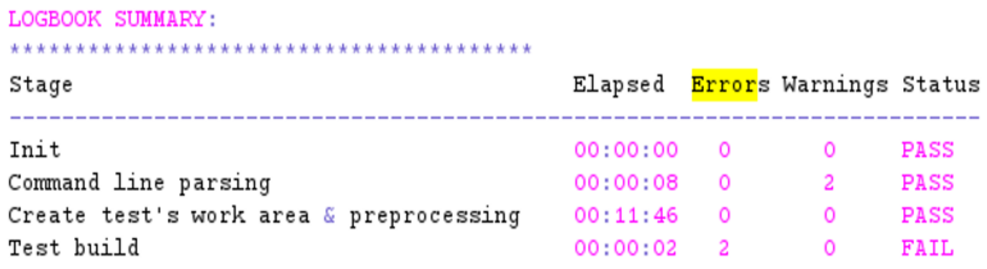
```
pmon_setup: disable counters; IA32_CR_PERF_GLOBAL_CTRL=0x000000000
pmon_setup: disable counters; IA32_CR_FIXED_CTR_CTRL=0x000000000
pmon_setup:resetting counter register P6_CR_PERFCTR0_wr: MSR addr=0xc1; dword_low=0x00000000; dword_hi=0x00000000; read_val=0x000000000
pmon_setup:resetting counter register P6_CR_PERFCTR1_wr: MSR addr=0xc2; dword_low=0x00000000; dword_hi=0x00000000; read_val=0x000000000
pmon_setup:resetting counter register P6_CR_PERFCTR2_wr: MSR addr=0xc3; dword_low=0x00000000; dword_hi=0x00000000; read_val=0x000000000
pmon_setup:resetting counter register P6_CR_PERFCTR3_wr: MSR addr=0xc4; dword_low=0x00000000; dword_hi=0x00000000; read_val=0x000000000
pmon_setup:resetting counter register P6_CR_PERFCTR4_wr: MSR addr=0xc5; dword_low=0x00000000; dword_hi=0x00000000; read_val=0x000000000
pmon_setup:resetting counter register P6_CR_PERFCTR5_wr: MSR addr=0xc6; dword_low=0x00000000; dword_hi=0x00000000; read_val=0x000000000
pmon_setup:resetting counter register P6_CR_PERFCTR6_wr: MSR addr=0xc7; dword_low=0x00000000; dword_hi=0x00000000; read_val=0x000000000
pmon_setup:resetting counter register P6_CR_PERFCTR7_wr: MSR addr=0xc8; dword_low=0x00000000; dword_hi=0x00000000; read_val=0x000000000
pmon_setup:resetting counter register IA32_CR_FIXED_CTR0_wr: MSR addr=0x309; dword_low=0x00000000; dword_hi=0x00000000; read_val=0x000000000
pmon_setup: PMON Event configured by writing register P6_CR_EVNTSEL0 : MSR Addr= 0x186; written_value=0x004307c0; read_value=0x0004307C0
pmon_setup: PMON Event configured by writing register P6_CR_EVNTSEL1 : MSR Addr= 0x187; written_value=0x004307c0; read_value=0x0004307C0
pmon_setup: PMON Event configured by writing register P6_CR_EVNTSEL2 : MSR Addr= 0x188; written_value=0x004307c0; read_value=0x0004307C0
pmon_setup: PMON Event configured by writing register P6_CR_EVNTSEL3 : MSR Addr= 0x189; written_value=0x004307c0; read_value=0x0004307C0
pmon_setup: PMON Event configured by writing register P6_CR_EVNTSEL4 : MSR Addr= 0x18a; written_value=0x004307c0; read_value=0x0004307C0
pmon_setup: PMON Event configured by writing register P6_CR_EVNTSEL5 : MSR Addr= 0x18b; written_value=0x004307c0; read_value=0x0004307C0
pmon_setup: PMON Event configured by writing register P6_CR_EVNTSEL6 : MSR Addr= 0x18c; written_value=0x004307c0; read_value=0x0004307C0
pmon_setup: PMON Event configured by writing register P6_CR_EVNTSEL7 : MSR Addr= 0x18d; written_value=0x004307c0; read_value=0x0004307C0
pmon_setup: Enable fixed Perfmon counters0; IA32_CR_FIXED_CTR_CTRL=0x111100000009
pmon_setup: globally enable fixed Perfmon counters; IA32_CR_PERF_GLOBAL_CTRL=0xF000000FF
```

Figure 5.2.2: Configuring event select Registers for upcoming event

It is shown in figure 5.2.2 that performance monitoring global counters and fixed counters are disabled, so its value is zero. After that local counters are reset to zero. This indicates that the value stored in counter register is zero.After resetting counters, event select registers are configured for the events that is to be monitored. After that local fixed PMON counters are enabled and at last global fixed counter is enabled.

5.3 Error Generation and Challanges

Result Analysis 1



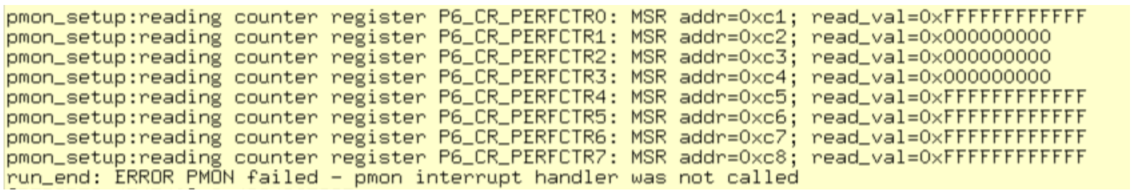
```
LOGBOOK SUMMARY:
****************************************
Stage                                      Elapsed   Errors Warnings Status
---------------------------------------------------------------------------
Init                                       00:00:00   0        0      PASS
Command line parsing                       00:00:08   0        2      PASS
Create test's work area & preprocessing    00:11:46   0        0      PASS
Test build                                 00:00:02   2        0      FAIL
```

Figure 5.3.1: Logbook summary of failing Log

As shown in the figure 5.3.1, the test fails in the build stage. This is due to environment failure. If compiling the test case is not successful then test fails in this stage.

Result Analysis 2

When the counter overflows, it should generate PMI request to provide handle and



```
pmon_setup:reading counter register P6_CR_PERFCTR0: MSR addr=0xc1; read_val=0xFFFFFFFFFFFF
pmon_setup:reading counter register P6_CR_PERFCTR1: MSR addr=0xc2; read_val=0x000000000
pmon_setup:reading counter register P6_CR_PERFCTR2: MSR addr=0xc3; read_val=0x000000000
pmon_setup:reading counter register P6_CR_PERFCTR3: MSR addr=0xc4; read_val=0x000000000
pmon_setup:reading counter register P6_CR_PERFCTR4: MSR addr=0xc5; read_val=0xFFFFFFFFFFFF
pmon_setup:reading counter register P6_CR_PERFCTR5: MSR addr=0xc6; read_val=0xFFFFFFFFFFFF
pmon_setup:reading counter register P6_CR_PERFCTR6: MSR addr=0xc7; read_val=0xFFFFFFFFFFFF
pmon_setup:reading counter register P6_CR_PERFCTR7: MSR addr=0xc8; read_val=0xFFFFFFFFFFFF
run_end: ERROR PMON failed - pmon interrupt handler was not called
```

Figure 5.3.2: PMON failed as handle was not called on PMI generation

counter sets to zero. But as shown in figure 5.3.2, interrupt handle was not called on PMI generation.

5.4 Conclusion

From this thesis work it is concluded that validating performance monitoring unit is significant for system to work more efficiently with high performance. Here counter facility with interrupt based approach and performance event based sampling (PEBS) method is used for performance monitoring at core and uncore level. To validate this at fullchip level is more important in order to have complete validation of DUT at core and uncore side. For validation, more effective test case and test environment is build to catch maximum bugs and make the system bug free.

5.5 Future Scope

For further enhancement in this work, coverage can be implemented for better verification.

References

[1] R. Arndt, E Levine, E. Sib, E. Welbon. “Performance Monitoring in Multiprocessor systems with Interrupt Masking,” filed as docket AT994181 in us.

[2] Charles Roth, Frank Levine, Ed Welbon IBM Corp, “Performance Monitoring on the PowerPCm 604 Microprocessor,” 11400 Burnet Rd. Austin, Texas 78758

[3] Jim Holt. “System-level Performance Verification of Multicore Systems-on-Chip”, 2009 10th International Workshop on Microprocessor Test and Verification, 12/2009

[4] Qiuming Luo1, Chang Kong, Yuanyuan Zhou, Guoqiang Liu. “Understanding the Data Trafc of Uncore in Westmere NUMA Architecture, 2014 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing.

[5] Intel 64 and IA-32 Architectures Software Developers Manual

[6] http://download.intel.com/support/processors/pentiummmx/sb/24318504.pdf