Pre-Silicon Validation of "Performance Monitoring Counters"

A Thesis Submitted in Partial Fulfillment of the Requirement for the Award of the Degree of

MASTER OF TECHNOLOGY

in VLSI Design

Submitted By

Monika

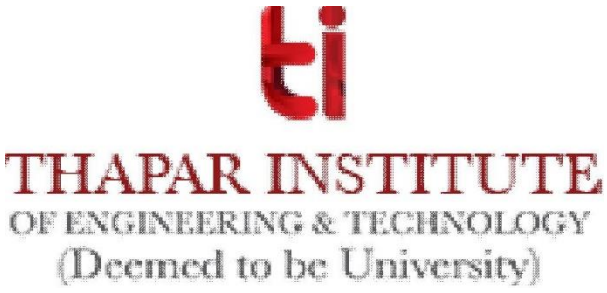
601662011

Under Supervision of

Mr. Sandeep Agrawal
Engineering Manager
Intel Technology India Pvt. Ltd. Bangalore

and

Dr. Mayank Kumar Rai
Associate Professor,ECED,TIET



ELECTRONICS AND COMMUNICATION ENGINEERING DEPARTMENT

THAPAR INSTITUTE OF ENGINEERING & TECHNOLOGY (A DEEMED TO BE UNIVERSITY), PATIALA, PUNJAB JUNE, 2018

INTEL TECHNOLOGY INDIA PVT. LTD.
Bangalore – 560103, Karnataka, India

Date: June 11, 2018

CERTIFICATE

This is to certify that **Monika Chauhan (Regn. No. 601662011)**, a student of M.Tech.(VLSI Design), **Thapar Institute of Engineering and Technology, Patiala** has successfully completed one year (August 2017 – June 2018) internship programme in **Intel Technology India Pvt. Ltd, Bangalore**. Her title of dissertation is "**Pre-silicon Validation of Performance Monitoring Counters**".

During the period of her internship programme, she was punctual and hardworking.

I wish her every success in life.

Sandeep Agrawal
Engineering Manager
Big CoreAV

INTEL TECHNOLOGY INDIA PVT. LTD. SRR 3 ★ 103 - BENGALURU

DECLARATION

I, Monika, hereby declare that the work presented in this thesis entitled "Pre-silicon Validation of Performance Monitoring Counters" in partial fulfillment of the requirement for the award of degree of Master of Technology (VLSI Design) submitted at ECED, Thapar Institute of Engineering and Technology (Deemed to be University), Patiala, is an authentic record of work carried out under supervision of Dr. Mayank Kumar Rai, Associate Professor, ECED, Thapar Institute of Engineering and Technology, from June 2017 to June 2018. The matter presented in this has not been submitted either in part or full to any other university or institute, for the award of any other degree.

Date: 18/06/2018

Monika

601662011

It is certified that the above statement made by the student is best to my knowledge and belief.

Date: 18/06/18

Dr. Mayank Kumar rai

Associate Professor

Dr. MAYANK KUMAR RAI
Associate Professor
Electronics & Communication Engg. Deptt.
Thapar Institute of Engg. & Technology
PATIALA-147004 PUNJAB (INDIA)

ACKNOWLEDGEMENT

I would like to express my sincere thanks to Dr. Alpana Agarwal **,**Head of Department, ECED, Thapar Institute of Engineering and Technology (Deemed to be University), Patiala, for giving me the opportunity to do a yearlong project in Intel India Pvt. Ltd. I owe my profound gratitude to Dr. Mayank Kumar Rai, Associate Professor, ECED, Thapar Institute of Engineering and Technology (Deemed to be University), Patiala, for his immense knowledge, continuous guidance and support. I express my sincere thanks to Mr. Sandeep Agrawal (Engineering Manager), Mr. Karthik Varadarajan (Technical Verification Lead), Intel Pvt. Ltd., who in spite of being extremely busy with their schedules, took out time to guide and help me with my project. I am also grateful to my colleagues for providing an amicable work environment.

Finally, I would also like to thank my family and friends for their patience and unwavering support during the course of this project.

Monika

(601662011)

ABSTRACT

Pre-silicon validation/verification is required to find the bug early in the system. They are easy to find and fix also cheaper to resolve the issue. Performance monitoring unit is a hardware unit which is built inside the CPUs of the X86 family and other architecture families like PowerPC and ARM. The PMONs store the counts of hardware related exercises inside the CPU, and can be utilized to analyze different parts of utilizations running on that hardware. The primary utilization of PMONs is software performance analysis.

Performance Validation is being done by programming programmable PMCs with architectural events like number of instructions decoded, number of interrupts received, number of cache miss and a lot more. Intel introduced Performance Monitoring in early Pentium processors with a set of two 40-bit Performance Monitoring Counters and with a set of model-specific performance monitoring counter MSRs. These counters permit selection of processor performance parameters to be monitored and measured. The information obtained from these counters can be used for tuning system, compiler performance and application performance.

The motivation behind this project is based on the expectation of the customer for higher performance on latest products. Performance validation at core level gives customers to do depth analysis of generation to generation architecture features and performance parameters. Hardware performance counters helps to obtain the micro-architecture level information. After obtaining the information the bottlenecks of the system are identified and removed which helps to improve application performance. The main goal of this project is running regression and checking the failing test and debugging them.

TABLE OF CONTENT

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

FPU: Floating Point Unit

FDIV: Floating Point Division

MAS: Micro-Architecture specification

HDL: Hardware Description Language

RTL: Register Transfer Level

DUT: Design under Test

PCB: Printed Circuit Board

DV: Dynamic Verification

FV: Formal Verification

MCU: Microcontroller Unit

OOO: Out-of-order

ALU: Arithmetic logic unit

PMU: Performance Monitoring Unit

PMON: Performance Monitor

PMC: Performance Monitoring Counter

MSR: Model Specific Register

TLB: Translocation Buffer

CHAPTER 1

INTRODUCTION AND STATEMENT OF THE PROBLEMS

Validation is a process in which a design is tested before the tape out and checks the correctness of the design. It is intended to check that a product or system should meet the requirement of users and behave according to the design specifications. First in the development phase, the process involves providing set of tests to model and simulate a portion or entirety of the design, then do a review or analysis of the modeling results. The post-development phase, validation process is to ensure that the product or services meeting the initial design requirements and specifications with the time. It plays a very important role in the VLSI industry. Validation requires a lot of time and huge manual effort. Now test time in validation plays a crucial role in time to market. So reducing the test time and making early time to market is an important goal of a validation engineer.

1.1 PURPOSE

Pre-Silicon verification is a process which involves multiple operations and is aimed at fulfilling the designer's objective if a design sticks to its specifications. Verification helps in reducing the number of bugs to a minimum possible value, when making the design. Verification is an important phase in developing the design. Without verification, one cannot provide an assurance that the design works well in all conditions. If the design is provided for tape out without any verification, then the probability of working on the chip will be very less, which leads to wastage of money and precious time. Pre-Silicon verification minimizes the cost required for testing. As the design reaches to the ending stage, the cost of verification increases. To assure that a chip is working properly, it is mandatory that each and every design is verified.

1.1.1 CPU Bugs

- **Meltdown and Spectre flaws:-**

  This is the catastrophe CPU bug that affects all the component of the system. This bug can affect our tablets, modern computer, smart phone and PC from all vendors that running on any operating system. The Meltdown and Spectre flaw has been found by security researchers at Google's Project Zero in association with many academic and industrial researchers from several countries. Meltdown and Spectre bug is a serious security flaw found in processors designed by Intel, ARM,AMD which could led the hackers steal sensitive information like your password and banking information. It could enable programmers to sidestep the equipment hindrance between applications keep running by clients and the PC's center

memory. These bugs are harder to fix. Fixing these bugs would need much experience and effort without impact on the performance of the system.

- **Pentium Bug:-**

  There was a bug called Pentium FDIV bug in the Intel Pentium Floating Point Unit (FPU). As a result of the FDIV bug, the processor was returning off base decimal results, this issue caused such huge numbers of inconveniences for the exact counts required in math also, science. This was experienced seldom by the clients (One of the magazines evaluated that roughly 1 out of 9000 million floating point separating would have delivered the wrong yield). So Intel reviewed all the blemished processors and supplanted them with new processors. It brought about lost 500 million dollars to Intel. The likelihood of finding the bug with the assistance of ordinary dynamic verification is less and the misfortune brought about not finding the bug is high. New technique called Formal verification is presented where whole state space is secured. So Formal approval has turned out to be overwhelming in each plan. In planning a processor, the imperative target is to dispense with every single bug in the framework. So verification has turned into a basic angle in outlining a framework.

1.2 TYPES OF VALIDATION PROCESS

Based on the validation done on the RTL and real silicon the validation can be classified into two types i.e. Pre-Si Validation and Post-Si Validation as mentioned in below figure.

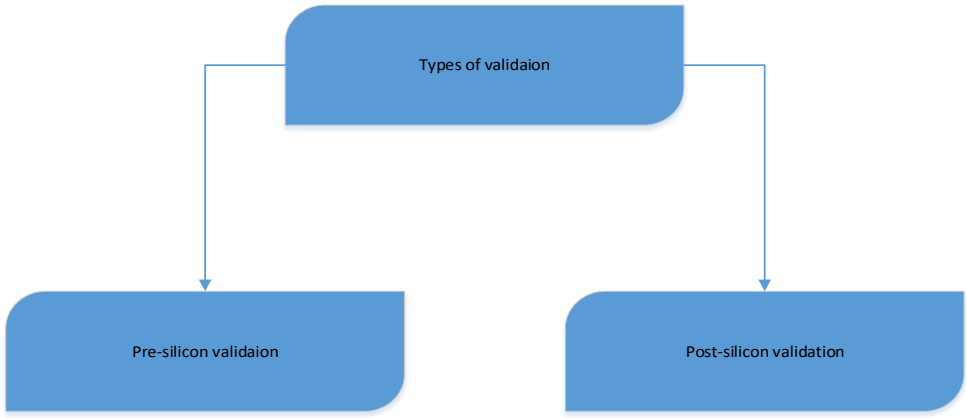


Figure 1.1Types of validation process.

1.2.1 Pre-Si Validation

Earlier the bugs found in the process are easy to resolve the issue. Also, there is a cost reduction for fixing a bug in pre-silicon which means they are cheaper to fix. The Figure 1.2 shows the variance of cost with design stages. Each and every functionality of the product is validated even the inputs which could cause an output error response. In most of the cases pre-silicon validation is executed at a chip, multi-chip or system level. Modelling of a full system must be done for this approach.

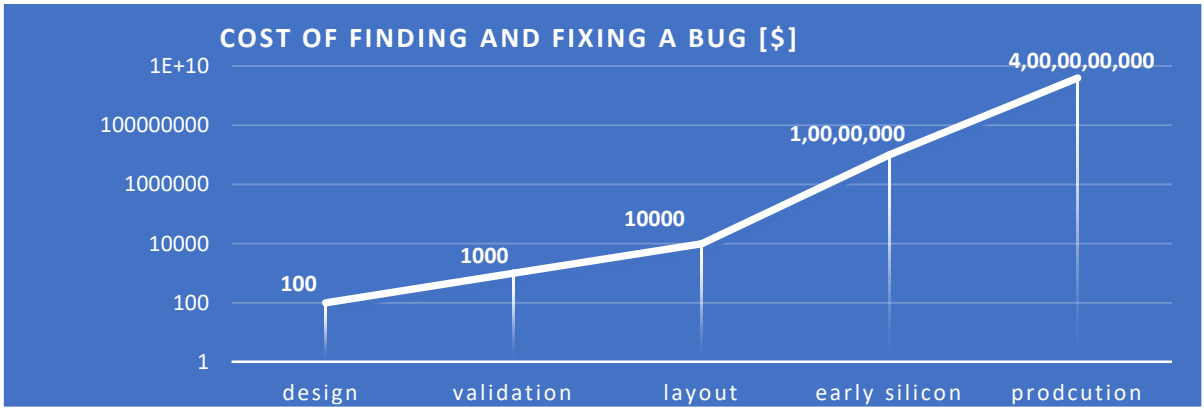


Figure 1.2 Design stage versus cost[11].

1.2.2. Post-Si Validation

Once the chip in silicon is out, validation is done to ensure that there are no shorts, inconsistency or any noise etc. It essentially refers to the real time testing of the silicon chip. Validation in real time environment usually runs at GHz frequency. Various steps involved in post-Si validation are:

- Detecting the problem by running test cases ranging from the instruction sequences run on pre-Si tests to end-user applications
- Locating and root-causing the problem based on the failure signature.
- Fixing the issue by applying patches or modifying the RTL design before next silicon step's tape in.

1.3 VLSI DESIGN FLOW PROCESS

The VLSI chip configuration is for the most part performed in two phases, in particular Front end and Back end shown in Figure 1.3 and Figure 1.4 Front end manages the RTL coding of the outline specifications and verification of the coded plan. Back end manages the combination of RTL code into netlist, verification of the planning requirements, drawing the format and sending to creation to fabricate the chip.

1.3.1 Basic Design Flow

- **System specifications**: -

  First of all to make a product design spec is defined on the basis of the need of users. The high level representations of any system are system specifications. System specification depends upon the various factors like physical dimensions (size of the chip), performance, and functionality.

- **Architectural design**: -

  The basic architectural design decisions are made in this step includes, RISC vs CISC, the number of arithmetic logic unit and floating point unit, size of cache and number and structures of pipeline etc. The result of architectural configuration is a Micro-Architectural Specification (MAS). While MAS is a printed portrayal, Architect can precisely anticipate the performance, power and size of the outline in view of such a depiction.

- **Behavioral or Functional design: -**

  Internal structure is not specified, the behavior is specified in terms of input, output and timing diagrams of each unit. The result of functional design is timing diagram. The information leads to improve the overall design process and also reduces the complexity of subsequent phases.

- **Logic design: -**

  In this step the arithmetic operations, and logic operations of the design, control flow, word widths, register allocation, that represent the functional design are derived and tested. This is characterized as Register Transfer Level (RTL) that is expressed in a Hardware Description Language (HDL), such as VHDL or Verilog.

- **Circuit design: -**

  In this step a circuit is represented on the basis of a logic design. It shows the complete diagram of circuit elements cells, gates and transistors also the interconnection between these elements. This representation is known as netlist.

- **Physical design: -**

  In this step the netlist is converted into circuit layout. Each logic component is converted into a geometric representation.

- **Fabrication: -**The layout design is sent to fabrication.

- **Packaging and testing: -**

  The wafer is manufactured and diced into singular chips in a creation office. Each chip is then bundled and tried to guarantee that it meets all the plan particulars and that it capacities legitimately.

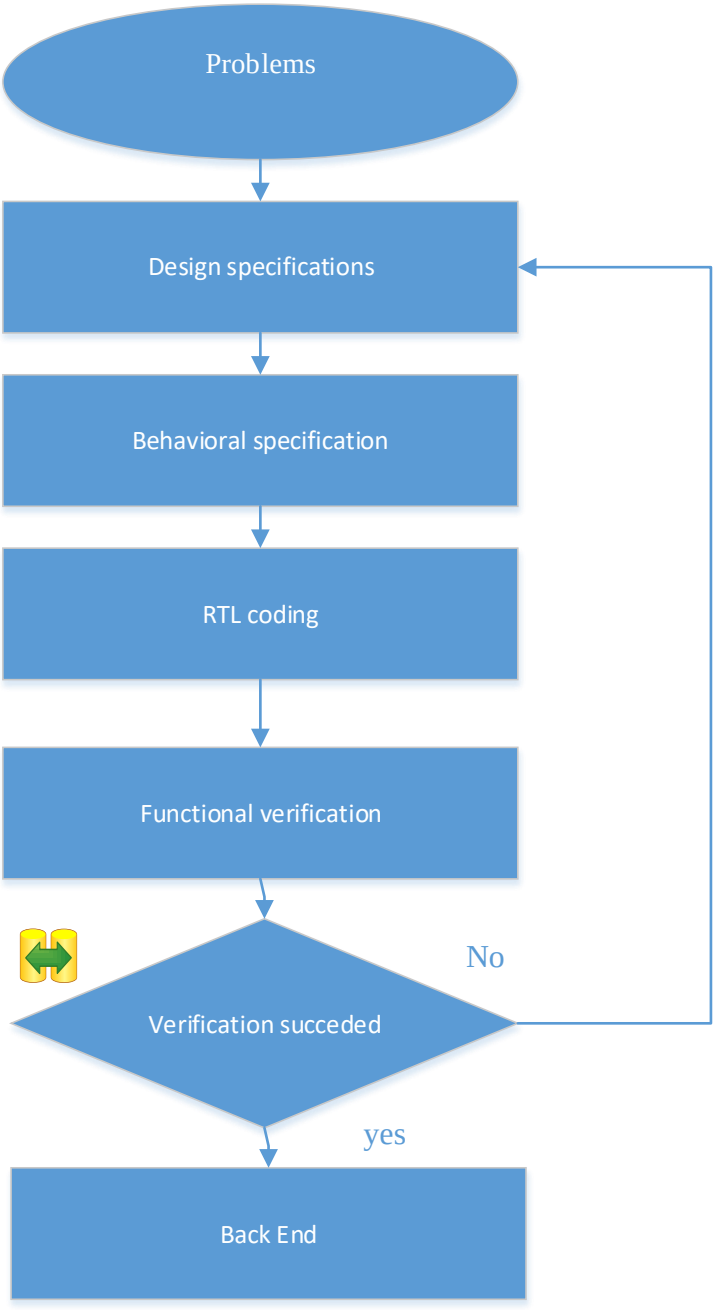


Figure1.3 Frontend flow.

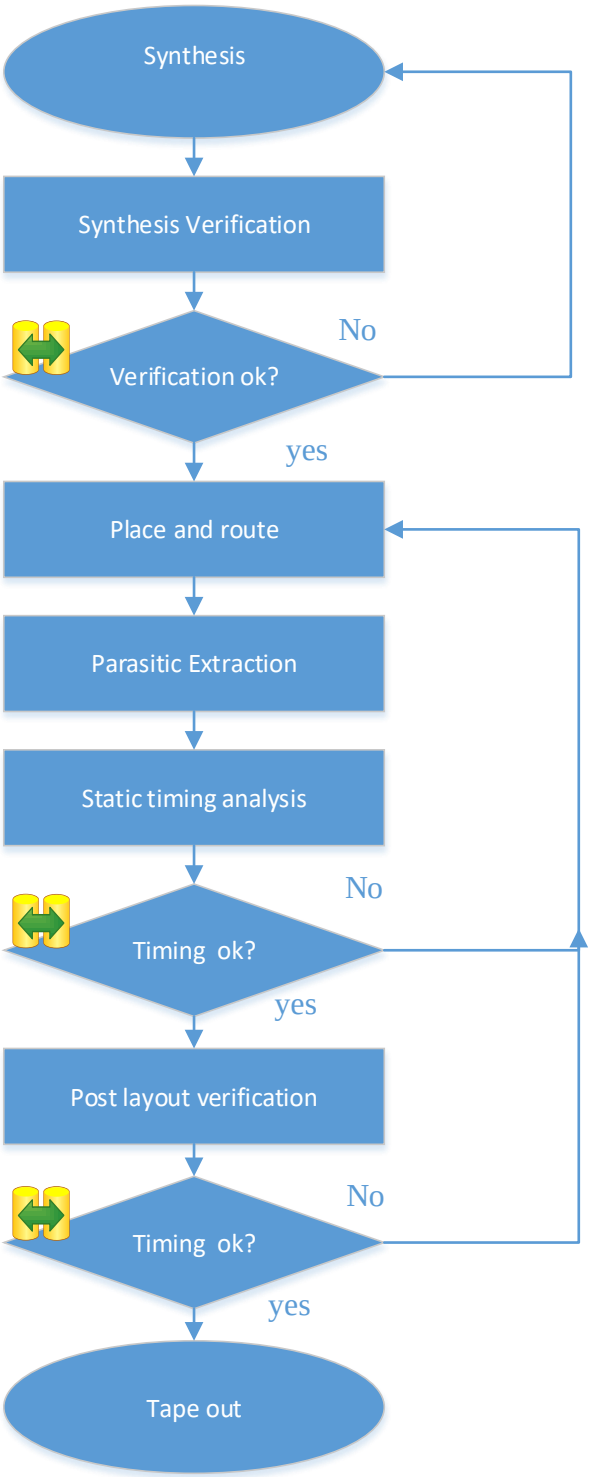


Figure1.4 Backend flow.

1.4 VALIDATION/VERIFICATION PROCESS

Validation or Verification is a process of checking that the design under test (DUT) behaves according to the specification defined. There is a need of understanding the microarchitecture specifications of the DUT to perform verification. All in all, whole outline will be separated into sub-blocks which might be additionally isolated into more sub squares. Small scale compositional specifications are the specifications of these sub hinders that should hold well after building up the outline. Verification is performed to build the accuracy in the outline. The chips must be verified practically before they get amassed onto the printed circuit boards. The decide of ten expresses that if a blame in a chip isn't found by chip testing then the cost of finding the blame expands 10 times at the printed circuit board level when contrasted with the chip level.

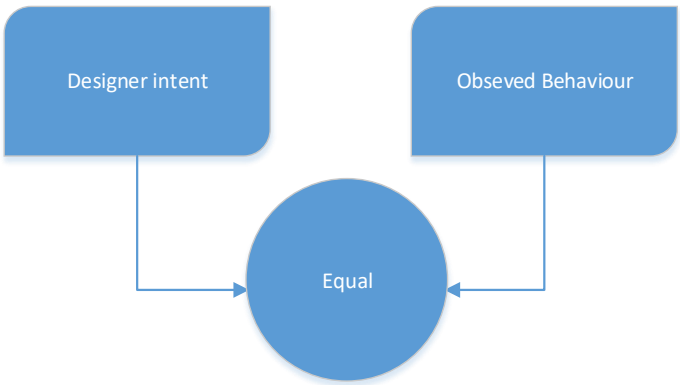


Figure 1.5 General validation behavior diag.

In the comparative way if the blame at PCB is not found at PCB level testing, at that point the cost of finding the blame increments by 10 times at framework level when contrasted with the PCB level. In the event that the shortcomings are not distinguished before tape in then the total creation cost is a misfortune. So verification is compulsory in VLSI chip outline. The block diagram of Validation process is shown in Figure 1.6

- **Validation**: - Testing a DUT under different test cases by comparing the output of DUT with the reference model. In general, stimuli generator provides the random inputs to the DUT and reference model. The output result of DUT is checked against reference model and units are compared via checker.
- **Validation Environment**: - Which supports the stimulating the DUT, checking the DUT, monitoring the DUT.
- **Checkers**: -The checker is one of the verification components which checks whether the yield from both the reference model and DUT are equivalent.
- **Monitor**: - Monitors the DUT during testing and reports the coverage of events.

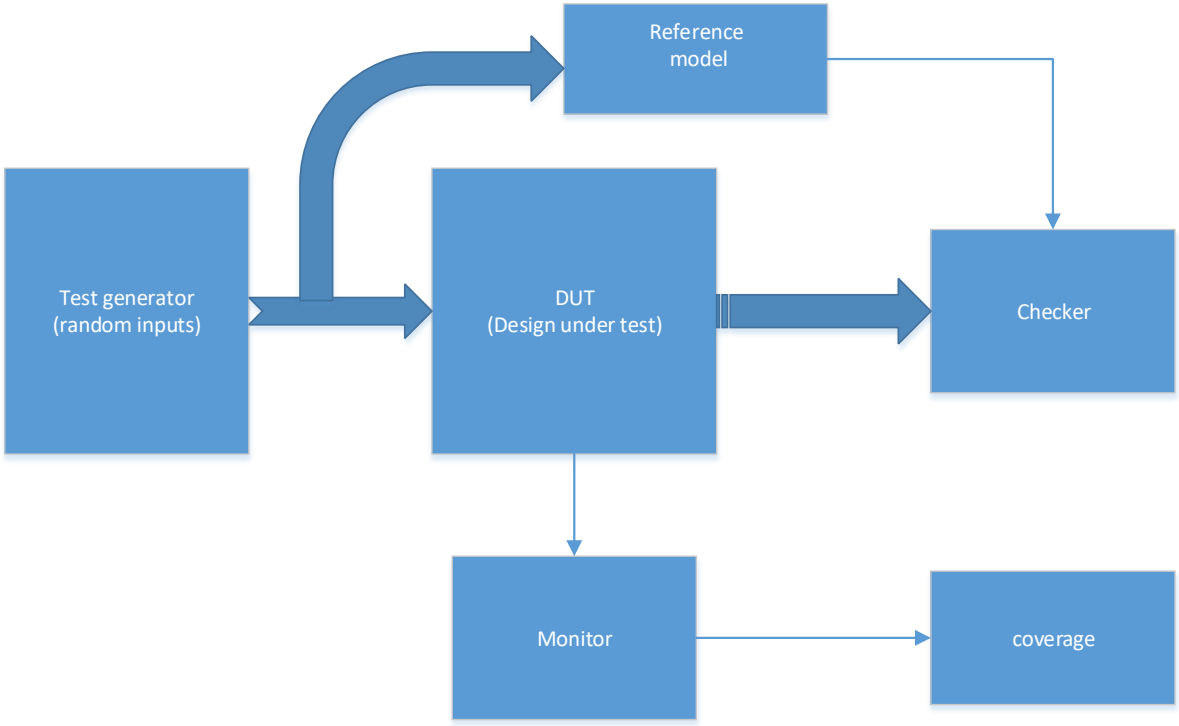


Figure 1.6 Validation Block Diagram.

a. Test generator

Test generators are producing arbitrary and additionally guided directions as stimuli to the reference display and DUT model to hit the usefulness and highlights according to determination, those guidelines are executed by a CPU core. The vast majorities of the test generators are composed in get together and furthermore get together with perl wrapper. Validators composes test such that some particular event is activated by a particular instruction. Execution particular test generators are composed to check new highlights usefulness, to watch some particular performance events, to program new counters, to include new piece fields in the programmable counter, to approve new bits, to intrude on the processor after some number of events, to report out bugs in RTL after some particular events tallies and so forth. The figure demonstrates the test condition capacities, for example, stimuli, scope and checking. Stimuli are the test contribution to practice DUT. Great stimuli rapidly practice the plan in least time. Coverage is utilized to distinguish gaps in the boost and exercise the DUT in some specific way. These capacities are between related. This figure clarifies there is more haphazardness that requirements more scope. More arbitrariness leads towards programmed checking.

b. Driver

The produced stimuli from the test generator aren't immediate contribution to the reference model and DUT. Driver is a specialist that drives stimuli from the test generator to the reference model and DUT in a fitting arrangement, since reference model and RTL does not straightforwardly take the created guideline from generator. Reference show takes just address and estimation of guideline in a particular arrangement, so this organization is given by the driver to drive the fitting boosts to the model. There are loads of switches and circuits in the driver part that drives produced guidelines in a fitting configuration to the model, creates vital log documents for troubleshoot reason.

c. Signal interface

The signal interfaces are used to send the output of DUT to the checker. The output of DUT is generally in the form of instructions from the pipeline. The information sent is in the format like load, store timings and globally observed timelines.

d. Checker

Checker in the validation condition checks and thinks about yield from the reference model and DUT. In the event that the data from RTL model and reference demonstrate same, at that point checker breeze through the test without bug in the environment. In the event that both not match, at that point test fails in run stage because of bug in the approval condition. The bug in the environment because of some genuine bug in RTL or because of bug in reference demonstrates. Checker is the fundamental specialist in the approval condition that checks usefulness, highlights and report out bugs from the environment.

e. Monitor

It monitors some particular intrigued flag and events for the execution reason. It checks some particular events from the RTL model that aides in execution counter examination with and without feature. In the event that there is substantial distinction in the counter an incentive with and without feature addition, that is known as performance bug. The revealed bug is fixed by the engineering and validator both, on the grounds that the bug because of expansion of feature in the processor. Monitor additionally assumes an imperative part in the execution approval since this operator gives a large portion of the execution parameters as number of events tally, IPC computation and so forth.

f. Trackers

Trackers additionally causes in execution approval to track some intrigued events on per-bit per thread premise with include and without feature. This kind of examination likewise find the execution issue and performance bugs. The component included the processor and their

overhead on the processor, chose by this kind of near examination from the tracker comes about.

g. Reference model

Reference demonstrates is the c - based reference model which is outlined by the different group. That group additionally actualizes the model concerning the detail. They show all feature as same as DUT, generally that will make bug in the environment because of reference model and test won't pass. Assume, RTL actualized feature yet that element by one means or another not demonstrated by the reference model group that makes reference model bug. Particular is same for the RTL and for the reference model however usage of the detail is unique.

Table 1.1 Level of validation comparison.

S.No.	High level	Low level
1.	Less controllability over stimuli	More controllability
2.	Slower operation	Faster
3.	Complex behavior	Simpler
4.	Dependent	Independent
5.	Less complex environment in validation	Complex
6.	External interface	Internal interface

Correlation at lower and more elevated amount approval condition

The table 1.1 shows how the lower level of approval helps the pre-silicon validation.

1.5 TYPES OF VERIFICATION

- Dynamic verification
- Formal verification

1.5.1 Dynamic Verification

Dynamic Verification (DV) is a method that is timing-simulation based verification of RTL model. This is the technique of verifying that the logic design behaves as per the specification. The stimulus generator generates and provides the input vectors to DUT that are used to search for abnormalities that exist between intent (specifications) and the implementation (RTL Code). The stimuli applied to design can be

- Random
- Direct

Large input stimuli can be covered by using directed random which generates stimuli randomly using some constraints or specifications. The outline of how dynamic verification works are shown in figure 1.7.

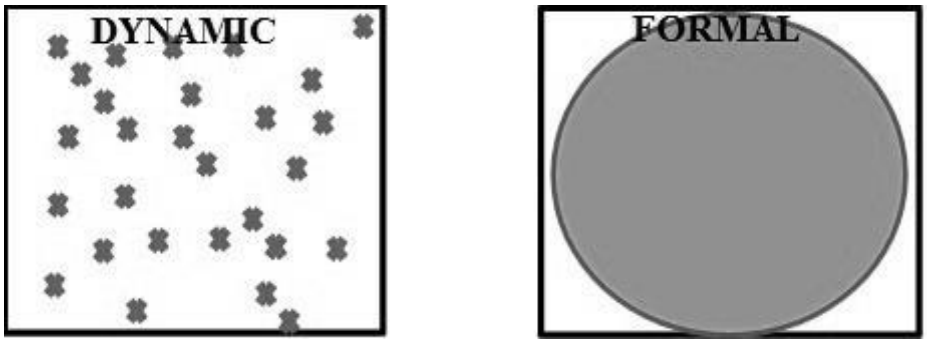


Figure 1.7 Dynamic and formal verification.

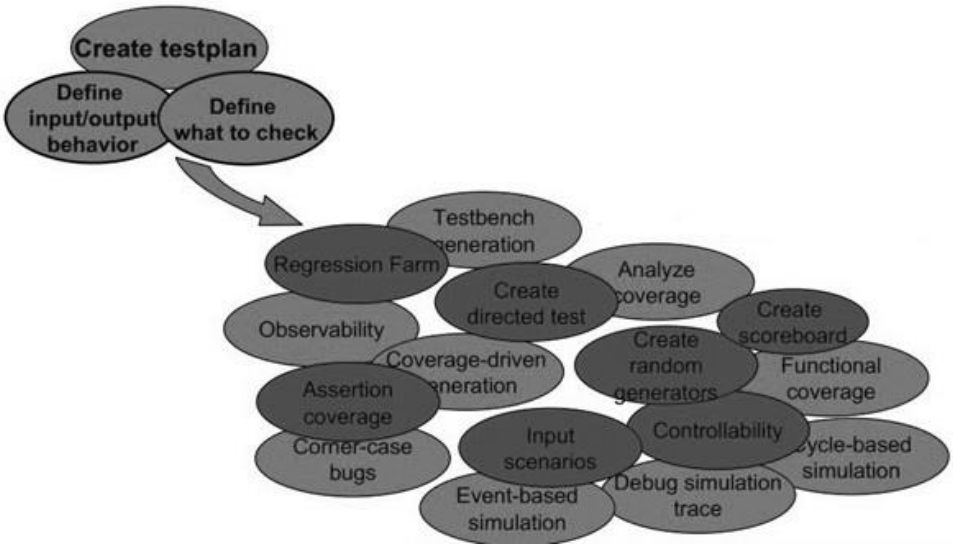


Figure 1.8 Dynamic Verification Scenario.

1.5.2 Formal Verification

Formal Verification (FV) is set of techniques that numerically demonstrate the characteristics of an outline, as opposed to dynamic verification (which just checks specific input values).

It is an approach to make the confirmation recommending the framework that it is possible that it does or does not have any property. It analyzes the formal model of the framework to a formal

specification of the property. It utilizes coherent induction to demonstrate that the model does or doesn't meet the specifications.

The outline aim is communicated as models that must be demonstrated in the implementation.

The different kinds of formal verification are

- Formal property verification
- Formal equivalence verification

Formal property verification

• Formal Verification

An assertion is an indication to the verification tool to check that a certain property or rule must hold true. Assertions produce bugs when events which are undesired are occurring. Assertions are coded using a formal language. The simulator is the one which checks the assertions and flags an error that it is not satisfied

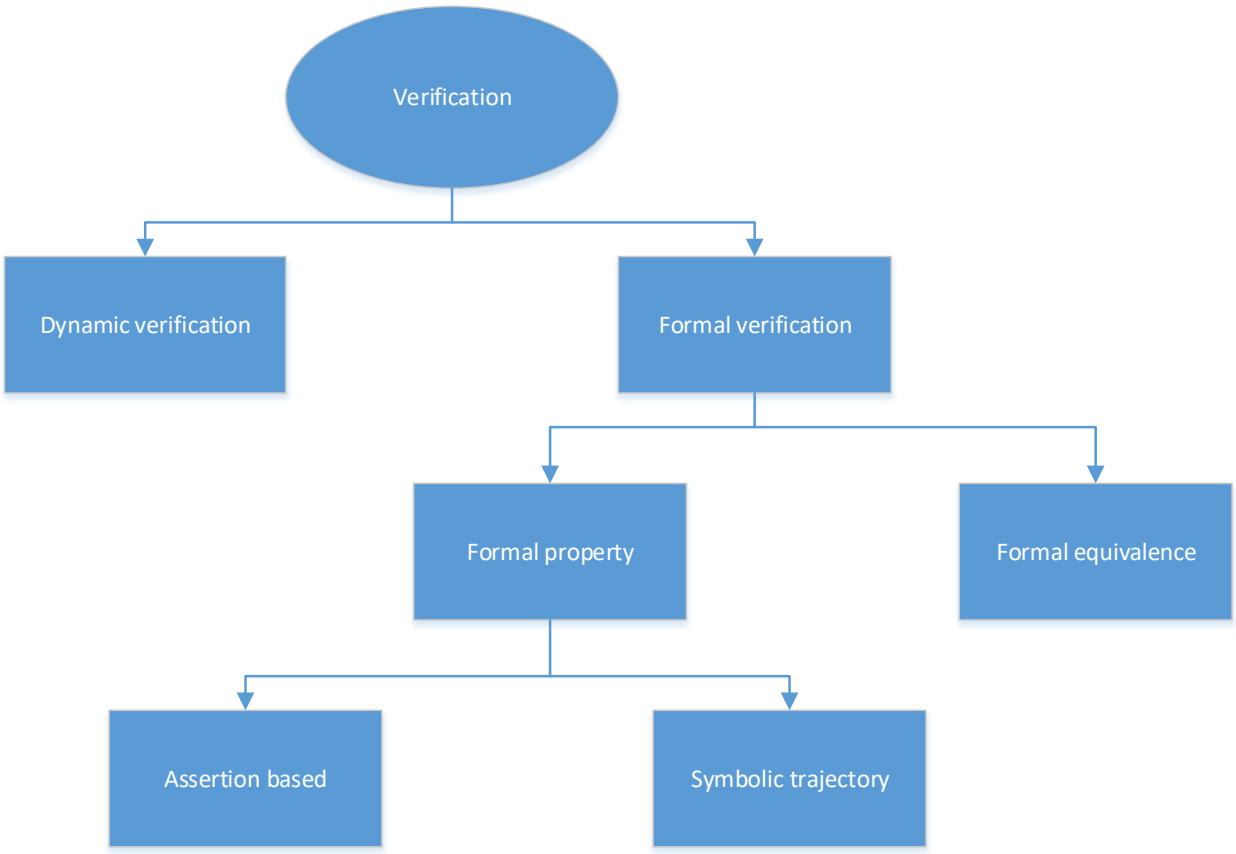


Figure 1.9 Formal Verification.

An assertion should be triggered only during particular events and care should be taken in triggering an assertion. Else it may give a false pass even though the assertion is not actually getting triggered. The tool helps in proving whether the property holds good or finds a counter

example that violates the property. The tool checks all the possible conditions that can violate the property. The input constraints are provided as assumptions by the designer. Verification at cluster or unit level is carried, depending on the design size and complexity.

• Symbolic Trajectory Evaluation

Symbolic trajectory evaluation (STE) is a formal verification method that uses a symbolic representation of three valued signals (typically using a BDD representation) to perform symbolic simulation (typically at the logical gate level) to verify that system's behavior meets a specification which is expressed in a confined temporal logic.

• Formal Equivalence Evaluation

Formal Equivalence Verification is a verification method which is used for proving that two models are equivalent. This can be used to prove, for example, that a circuit intended to work equivalent to the RTL is working good. It compares specification and implementation.

1.6 PIPELINE AND OUT OF ORDER EXECUTION

Pipelines are a fundamental component in present day processor microarchitecture. Indeed, even RISC processors cannot execute an instruction in a single clock cycle without a pipelined outline. The pipeline breaks instruction decodes unit and execution unit into several parts. So that during each cycle processor can execute the instructions. Yet, microcontroller (MCU) originators must take care to adjust the profundity of the pipeline. Also the group must choose how much silicon to commit to pipeline preparing in light of the fact that kick the bucket measure relates straightforwardly to IC cost.

1.6.1 OOO Execution

The PMU unit resides inside OOO unit in a processor. The various steps include in an out of order execution are

- Instruction fetch: - Instructions are issued sequentially by a processor regardless that executed in a non-sequential manner.
- Instruction decode: - This unit reads the instructions from instruction queue and decodes them.
- Instruction dispatch: - This unit uses reservation stations also known as instruction buffer. It can dispatch instructions to functional unit in a parallel manner.
- The instructions are executed in out of order. Until the input operands of the instructions are available instruction awaits in the queue. As operands get available they are allowed to leave before older instructions.

- There are multiple functional units. Appropriate functional unit is used to execute the issued instruction.
- The results are queued.
- Commit or retire stage.

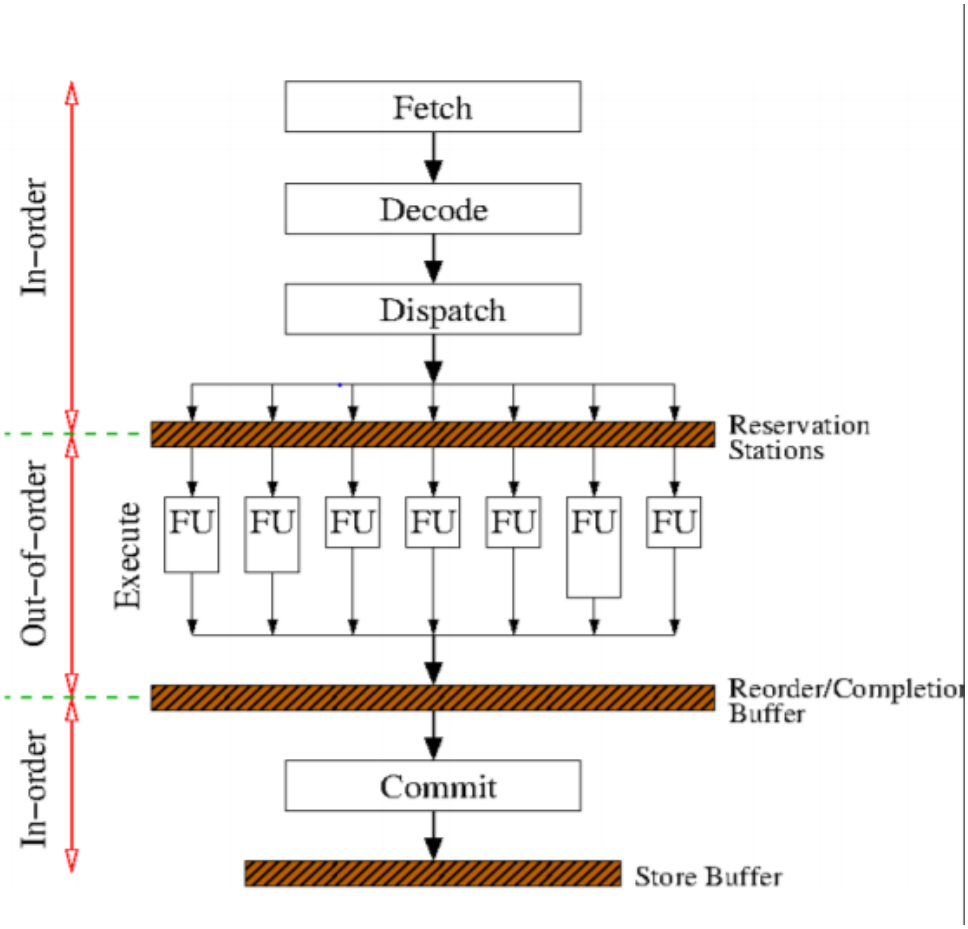


Figure 1.10 OOO Execution flow [21].

Conflicts which arise during executions and prevent instruction to be executed are:

- Resource conflicts.
- Data dependencies.

(a) Resource conflicts:

This conflict arises during execution if more than one instruction requires the same hardware resource at same time. A resource refers to a register, memory or ALU. Several instructions compete for the same hardware resource at the same time.

e.g., two arithmetic instructions need the same floating-point unit for execution. This is similar to structural hazards in pipeline.

They can be solved partly by introducing several hardware units for the same functions.

e.g., have two floating-point units.

The hardware units can also be pipelined to support several operations at the same time.

(b) Data dependencies:

The processor executes many instructions simultaneously. When data used by an instruction depends upon the data produced by other which has not produced yet then these kinds of dependencies can affect the execution time.

True Dependency

True data dependencies exist when the output of one instruction is required as an input to a subsequent instruction:

MUL R3,R2,R1 (R3 := R2 * R1)

ADD R5,R3,R4 (R5 := R3 + R4)

The second instruction can be fetched and decode in parallel with first. But it cannot execute till the first instruction gave the results and retire. These intrinsic features of the user's program cannot be handled by hardware techniques.

Output Dependency

If two instructions are writing into the same memory location at the same clock cycle then output dependency exists.

If the second instruction writes before the first one, an error occurs:

MUL R4,R3,R1 (R4 := R3 * R1)

ADD R4,R2,R5 (R4 := R2 + R5)

Anti-Dependency

An anti-dependency exists if an instruction uses a location as an operand while a following one is writing into that location.

If the first one is still using the location when the second one writes into it, an error occurs:

MUL R4,R3,R1 (R4 := R3 * R1)

ADD R3,R2,R5 (R3 := R2 + R5)

Elimination

Output dependencies and anti-dependencies can usually be eliminated by using additional registers. This technique is called register renaming.

MUL R4,R3,R1 (R4 := R3 * R1)

ADD R9,R2,R5 (R9 := R2 + R5)

MUL R4,R3,R1 (R4 := R3 * R1)

ADD R9,R2,R5 (R9 := R2 + R5)

Example:

WAR dependency exist between LD R7, (R3) and SUB R3, R12, R11 instructions

Before Renaming	After Renaming
ADD R3, R4, R5	ADD hw3, hw4, hw5
LD R7, (R3)	LD hw7, (hw3)
SUB R3, R12, R11	SUB hw20, hw12, hw11
ST (R15), R3	ST (hw15), hw20

With Register Renaming, the first write to R3 maps to hw3, while the second write maps to hw20.This converts four instruction dependency chain into 2 two instructions chains, which can then be executed in parallel if the processor allows out of order execution.

1.7 STATEMENT OF THE PROBLEMS

The objectives of proposed works in the thesis are briefly as follows:

- Understanding the concepts of Pre-Si validation.
- Understanding various concepts related to Performance Monitoring Validation.
- Launching the various regressions to enable the perfmon in the model. Checking the performance events count in both behavioral and RTL model.
- Addition of the new Perfmetrics events to improve the performance in the next core model.
- Checking and debugging the failures in both the models.
- Tools enhancement for checking new performance events in the core.

1.8 ORGANISATION OF THESIS

In **Chapter 2,** it is explained that, what is the use of performance monitoring unit in the latest processor? The various methodology used for the measuring the performance monitoring counters accuracy. Two modes of the performance monitoring hardware counting mode and sampling mode were discussed. It tells about the exactness of performance monitoring hardware and how accurate is this to use. The basic performance bottlenecks in the CPU was also determined by top-down approach

In **Chapter 3,** the performance monitoring unit is explained briefly. PMU unit is built inside the CPU to track the performance parameter of an application or a system. Different types of performance events are used to obtain the architectural and micro-architectural information of the system like cache hit or miss, branch misses, instruction cycles. This explains the PMU architecture and also model specific registers which are used to contain the information of PMON events. This also discuss

the reading and writing of the model specific registers to select and program distinct performance event.

In **Chapter 4,** this gives a prominent example is the non-linear CPU utilization on processors with Intel® Hyper-Threading Technology (Intel® HT Technology). Intel® HT technology is a great performance feature that can boost performance by up to 30%.Gives the detail about the Intel processors that they already have the capability to monitor performance parameter. The CPU resource utilization will depends upon the information obtained from the PMU.

In **Chapter 5,** results of the regressions are explained. Snippet lists are run to check the various test. Toolchain enhancement for checking new performance events in the core. The results of regression runs results in different work week and pass rate of the some events in the project. Perfmon has been enabled in the new model by toolchain. The toolchain enhancement for adding the new performance events to the new model to make performance more efficient.

In **Chapter 6,** the overall conclusion and future scope of the thesis is explained.

CHAPTER 2

LITERATURE SURVEY

2.1 INTRODUCTION

The performance monitoring unit is used to track the performance events. Performance monitoring counters are used to count the value of perf events. Various paper are studied which explains distinct methodology to determine the accuracy of PMC. The performance counters are widely used by application developers. Performance monitoring features gives the data that describes how an operating system and application behaves on the processor. This contains the information which can lead efforts us to enhance the performance.

W. Korn, P. Teller *et al.*[1], explains methodology for measuring the accuracy of these performance monitoring counters. There is a wide utilization of performance counters by application developers gives prove that performance counters are certainly justified regardless of the silicon and configuration time required to incorporate them on present day microchips. These counters give simple execution estimations that could conceivably be precise.

This paper shows philosophy for deciding the precision of these counters and additionally preparatory after effects of an investigation that, utilizing this approach, assesses the exactness of the R12000 execution counters as for eight of 30 measurable events. The outcomes show that care must be taken when utilizing information created by performance counters in light of the fact that, at times, this information may prompt incorrect conclusions. This can happen when the granularity of the deliberate code isn't adequate to guarantee that the overhead presented by counter interfaces does not command the event count.

B. Sprunt[2],the performance monitoring features gives the data that describes how an operating system and application behaves on the processor. This contains the information which can lead efforts us to enhance the performance. Most current, elite processors have exceptional, on-chip equipment that screens processor performance. Information gathered by this hardware gives performance data on applications, the working framework, and the processor. This information can manage performance execution change endeavors by giving data that enables software engineers to tune the calculations utilized by the applications and working framework, and the code groupings that actualize those calculations.

Michael E. Maxwell, Patricia J. Teller *et al.*[3],discussed two modes of the performance monitoring hardware counting mode and sampling mode. It tells about the exactness of performance monitoring hardware and how accurate is this to use. As the performance monitoring hardware is built inside CPU having performance counters and other registers that are used to record data of the processor events.

In counting mode events are aggregated and accumulated. In sampling mode, the collection of data is based upon time- based or event-based sampling. This paper discusses uses of these two modes and considers the accuracy issues raised by each.

S. Eranian[4],this paper proposed a generic, flexible, and powerful monitoring interface to provide an access to the performance counters of modern processors designed. The interface helps performance tool to gather the count information. It provides us the overview of the interface. Also supports the major processor architecture. A performance monitoring tool understands the behavior of the applications and operating system. The collected information is utilized to enhance the performance in application, kernels, operating system, compilers and hardware. As processor implementation improvements shift from clock speed to multi-thread, multi-core, therefore need for accurate monitoring will arise significantly.The perfmon2 interface is very useful to address those needs.

S. Eyerman, L. Eeckhout, T. Karkhanis *et al.*[5],counter architecture, utilizing a diagnostic superscalar performance model, handles overlap effects more precisely than existing methods. Programmers can pick up understanding into software-hardware interactions by breaking down processor performance into individual cycles-per-instruction components that separate cycles consumed in dynamic calculation from those spent taking care of various miss events. Constructing accurate CPI components for out-of-order superscalar processors is complicated, however, because computation and miss event handling overlap.

Proposed hardware performance counter architecture and compared it with two simulation-derived CPI stacks. The simulation-derived CPI stacks serve as a reference for comparison. They use two simulation-derived stacks because of the difficulty in defining what a standard, correct CPI stack should look like. In particular, there might be cycles that one could reasonably ascribe to more than one miss event because of overlaps.

V. Weaver and S. McKee[6],when an architectural tool is created, it is essential to know that the output results are correct or not. A sanity check is done by comparing the tool's output with the hardware performance counters. For the exact result the output should match, if this does not happen than it indicates problems with the tool like unexpected behavior of the tool or real hardware.

This paper proposedthe behavior of the SPEC benchmarks against both dynamic binary instrumentation (DBI) tools and hardware counters. For distinct implementations of the x86 architecture, the retired instructions are calculated and result is collected in performance counters. When run with no special preparation, hardware counters have a coefficient of variation of up to 1.07%. After analyzing results in depth, find that minor changes to the experimental setup reduce observed errors to less than 0.002% for all benchmarks. The fact that subtle changes in how experiments are conducted can largely impact observed results is unexpected, and it is important that researchers using these counters be aware of the issues involved.

Jeff Diamond, John D. McCalpin, Martin Burtscher *et al*.[7],the modern supercomputers are having multicore processors to enhance the performance of such systems requires analysis and optimization techniques that specifically understand the multicore environment. This paper proposed the examination of legacy single core metrics and shows the misleading factors that can be carried out to multicore system, also examines the performance bottlenecks for multi-core system. This paperdemonstrates the performance challenges that can cause problems in multi-core system. The case study done is built upon the Intel Nehalem quad-core processors. The analysis leads the increase in performance by 35 percent.

Vincent M. Weaver, Dan Terpstra, Shirley Moore[8],investigates different techniques for working around the constraints of the x86 64 events, yet by and large this isn't conceivable and would require design upgrade of the basic PMU. Ideal hardware performance counters give correct deterministic outcomes. Genuine performance unit (PMU) usage doesn't generally satisfy this perfect. Events that ought to be correct and deterministic, (for example, retired instruction) demonstrate variation and over count on x86 64 machines, notwithstanding when keep running in entirely controlled conditions. Examination of eleven distinctive x86 64 CPU usage find the wellsprings of dissimilarity from expected check aggregates. Of all the counter events examination, find just a couple of that show enough determinism to be utilized without change in deterministic execution situations.

Ahmad Yasin[9],this paper exhibited Top-Down Analysis technique, systematic in-production investigation philosophy to distinguish basic performance bottlenecks in out-of-order CPUs. Utilizing assigned PMU events in multi-cores, the technique embraces a progressive arrangement, empowering the client to focus in on issues that straightforwardly prompt problematic performance. The strategy was exhibited to group basic bottlenecks, crosswise over assortment of customer and server workloads, with numerous microarchitectures' ages, and focusing on both single-threaded and multi-cores situations.

The bits of knowledge from this strategy are utilized to propose a novel ease performance counters architecture that can decide the genuine bottlenecks of a general out-of-order processor. Just eight new events are required. The essential point of performance monitoring units (PMUs) is to empower programming designers to adequately tune their workload for most extreme performance on a given framework. Current processors uncover several performance events, any of which could possibly identify with the bottlenecks of a specific workload. Stood up to with an immense volume of information, it is a challenge to decide the true bottlenecks out of these events.

Intelpedia[Intelpedia (n.d.)][12]**,**is an internal web tool used within Intel organization that provides the database for all the terms and concepts. It helps all the employees to have a quick reference for the concepts.

Pentium III Intel Architecture Software Developer's Manual[11],The Intel® 64 and IA-32 Architectures Software Developer's Manual, this manual contains all the basic architectural information and programming environment of Intel processors. It also has the information related to the instruction set of the processor and the structure of the opcode.

J.C. Saez, A. Pousa, R. Rodriíguez-Rodriíguez *et al.*[10], proposed PMCTrack tool which enables the OS scheduler. This is a tool that provides a simplest independent architecture mechanism. The OS scheduler can access per thread PMC data. In this paper the distinct case studies are analyzed that denote flexibility, simplicity and other powerful features of PMCTrack.Despite being an OS-oriented tool, PMCTrack still allows the gathering of monitoring data from userspace, enabling kernel developers to carry out the necessary offline analysis and debugging to assist them during the scheduler design process.

PMCTrack's monitoring modules enabled us to extensively evaluate (on real asymmetric hardware) PMCbased asymmetry-aware schedulers that were evaluated before using simulators [24] or using emulated asymmetric hardware [23]. More importantly, the implementation of these modules heavily relies on PMCTrack's in-kernel explicit event-multiplexing capabilities, which are not available in userspace-oriented PMC tools, such as perf [25].

CHAPTER3

PERFORMANCE MONITORING

3.1 INTRODUCTION

Performance monitoring tool is all about collection of information regarding how an application or system execute. Hardware performance counters helps to obtain the micro-architecture level information. After obtaining the information the bottlenecks of the system are identified and removed which helps to improve application performance.

Performance monitoring was presented in the X86 family associations with the Pentium processor. Having an arrangement of model specific registers (MSRs) utilized as performance counters. These counters allow a determination of processor performance parameters to be observed and estimated. As indicated by Intel writing the data acquired from these counters can be utilized for tuning framework and compiler performance [Int09]. Comparable highlights are presently accessible in the ARM [ARM] and PowerQUICC [Sem07] designs.

The PMON events measured value can be vary among different Intel CPU types. These comprises various CPU functions such as number of floating points operations, TLB misses, CPU ticks, branch predictor misses, and many more.

The PMONs can be divided into 2 main families:

1. Architectural PMONs – These PMON events will provide the fixed results on performing the same code over distinct machines of the architecture (e.g. all Atom CPUs with distinct memory, cache size etc.).
2. Micro-Architectural PMONs – These PMON events are dependent on specific hardware implementation of CPU and determine model specific functions (e.g., PMONs which candetermine the behavior of the cache, branch predictor etc.).

3.2 PERFORMANCE MONITORING UNIT (PMU)

Performance Monitoring Unit (PMU) is found in all high-end processors nowadays. The PMU is a hardware unit that is build inside out of order in a processor. This unit is used to determine the performance parameters of a processor. One can determine many performance parameters that depend upon the hardware provided by the processor and the feature it support. The performance parameters which can be measured are instruction cycles, cache hits, cache misses, branch misses etc. The PMU registers are configured and can be utilized to measure the performance parameters.

3.2.1. NEHALEM-BASED PMU ARCHITECTURE

Intel processor cores for a long time incorporated a Performance Monitoring Unit (PMU). This unit gave the capacity to check the micro-architectural events and count the occurrence an event which uncovers a portion of the internal workings of the processor core as it executes code. One use of this capacity is to make a rundown of performance events from which certain execution measurements can be ascertained. Programming designs the PMU to tally performance event over an interim of time and report the subsequent events count checks. Utilizing this strategy, performance analyst can portray general system performance.

Feature	Description	P4	Yonah (V1*)	Merom (V2*)	Penryn (V2*)	Nehalem (V3*)
Number of General Counters	Number of general counters per logical CPU.	18	2	2	2	4
Number of Fixed Counters	Examples include instructions retired, un-halted core clock ticks, un-halted reference clock ticks, etc.	0	0	3	3	3

Figure 3.1 Nehalem-based Performance Monitoring Counters [11].

3.3 PMU HARDWARE

The PMU in the Intel IA-32 design comprises of two sorts of registers or Model-Specific Registers (MSRs). Every processor model contains some registers distinct from the other models (even from a similar organization), that's why they are called as model specific register.

These two kinds are the Performance Monitoring Counters (PMC) and the Performance Event Select Registers. The performance monitoring counters (PMC) are used to count the events.

To determine a performance event, need to program the event select register.So, both the event selector register and PMC are essential to determine the performance event.

3.3.1. MODEL SPECIFIC REGISTERS (MSRs)

Performance monitoring related event MSRs allows the software to select some specific event from bundle of events. These registers are used in the performance monitoring unit counters and counter related control registers. These counters and control registers are accessed by the RDMSR and WRMSR instruction.

3.3.2. CONTROL AND STATUS REGISTER

Global control registers control the fixed and programmable counters and provides status indication in the status register.

3.3.2.1 IA32_PERF_GLOBAL_CTRL

The Perf_Global_Ctrl register is used to globally control the fixed as well as programmable counters. This register contains a control bit, if it is clear the counter will be ignored for all the other control register programming for specific counter. The counter will not count in that case.

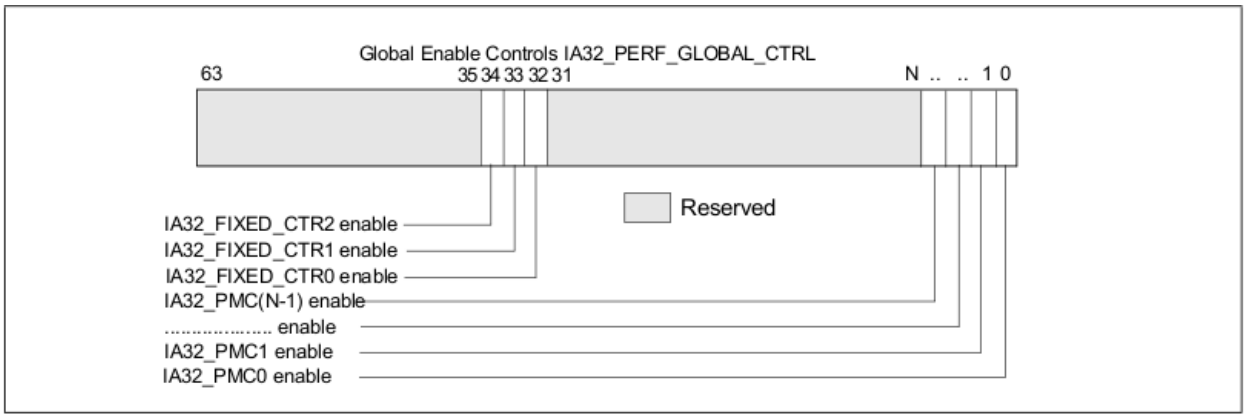


Figure 3.2 IA32_PERF_GLOBAL_CTRL MSR LAYOUT [11].

3.3.2.2 IA32_PERF_GLOBAL_STATUS

This register is used to indicate the overflow status of each of the fixed and programmable counters. The upper bits provide additional status information of the PerfMon facilities. If a bit is set this indicates an overflow has occurred in the corresponding counter. Overflow status bits in this register can be cleared by writing the IA32_PERF_GLOBAL_OVF_CTRL register.

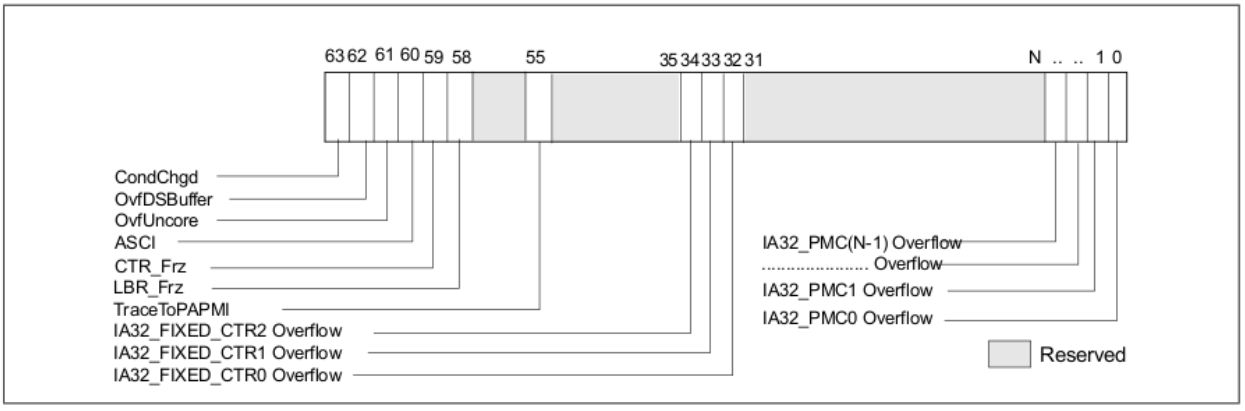


Figure 3.3 IA32_PERF_GLOBAL_STATUS MSR LAYOUT [11].

3.3.2.3 IA32_FIXED_CTR_CTRL

The fixed counters are controlled by this register. This register helps to determine whether these counters count in user or supervisor mode, or both. Also whether these counters are enabled to generate performance interrupts.

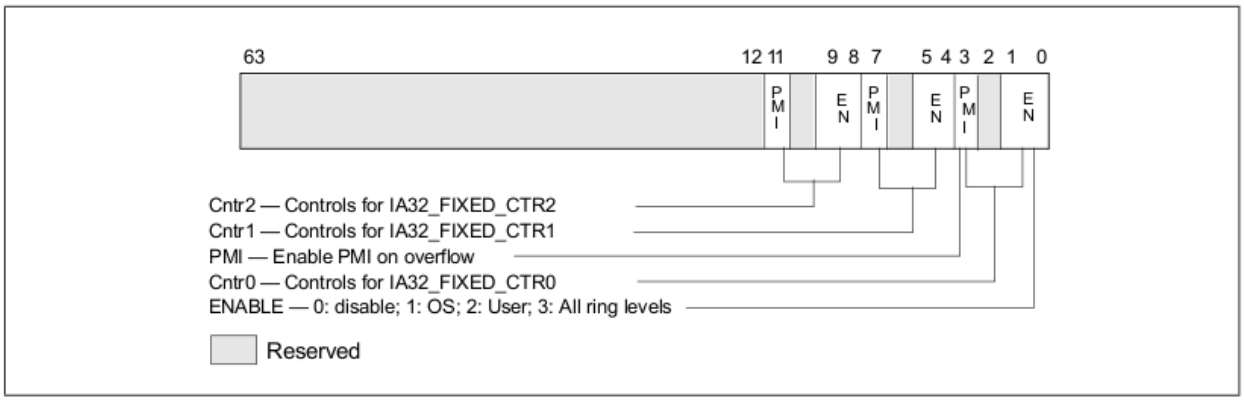


Figure 3.4 IA32_FIXED_CTR_CTRL MSR LAYOUT [11].

3.3.2.4 IA32_DEBUGCTL:

This register controls tracing, single stepping, and last branch record (LBR) collection, and certain freeze functions related to the PMU. The programming for this register is summarized in the figure below.

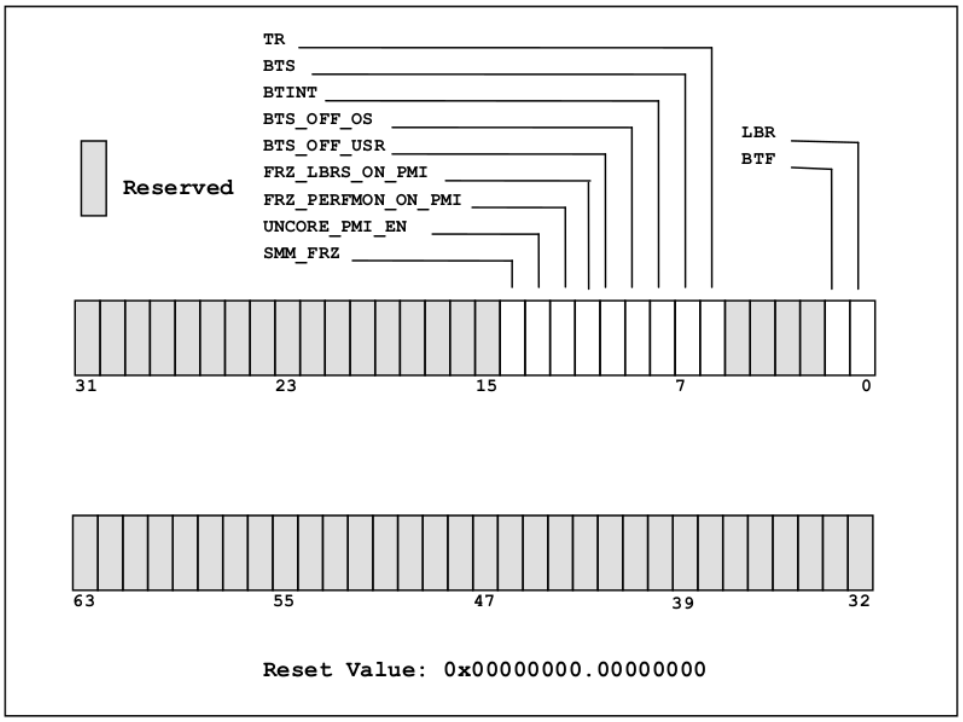


Figure3.5 IA32_DEBUGCTL MSR LAYOUT [11].

3.3.2.5 IA32_PerfEvtSelx

The PerfEvtSelX registers control the four programmable counters. Using these control registers, software can select the event to be counted, and the constraints under which those events are counted. Each counter must be locally enabled by this register, as well as globally enabled, in order to operate correctly.

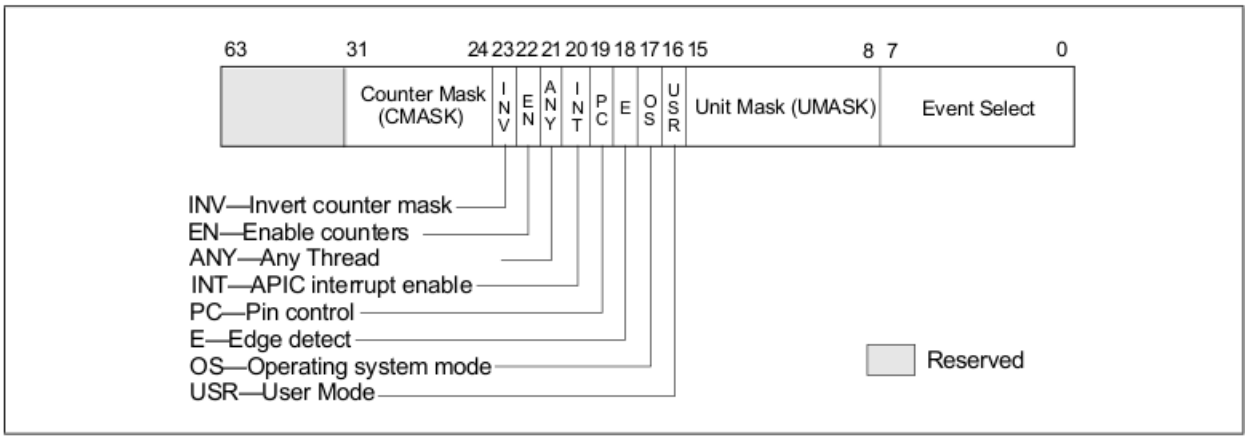


Figure 3.6 IA32_PerfEvtSelx MSR LAYOUT [11].

Performance event select registers can be programmed by using following values explained.

Event Select Fields (bits 0 to 7):

These bits are used to select a logic unit that detects a performance event to be monitored. These values are determined by computer architecture.

Unit mask (UMASK) fields (bits 8 to15):

There are multiple events which can be monitored by using the logic unit (selected in event select field). So the UMASK field selects those events which are to be monitor. This field provides us the one fixed value or multiple values that depend upon architecture.

USR flag (bit 16):

If this bit value is set, that means processor is running in user mode and the logic unit will monitor only those events which happened during user mode.

OS Flag (bit 17):

If this bit value is set, that means processor is running in highest privilege mode and the logic unit will monitor only those events which happened during OS mode.

The OS flag and the USR flag are used together to monitor or determine count of all the events.

E (Edge Detect) (bit 18):

If set, It is used when selected events goes from low to high state only then the counter will increment the value.

PC (Pin Control) (bit 19):

If the bit is reset, toggles the PMI pin when the counter overflows. If the pin is set, it will increment the counter along with toggling the PMI if event occurs.

INT (APIC interrupt enable) flag (bit 20):

If set, when the performance monitoring counter overflows the processor is used to raise an interrupt.

EN (Enable Counters) flag (bit 22):

If reset, disables the counters. If set, it enables the performance monitoring counters for an event.

INV (inversion) flag (bit 23):

If set, used to invert the output of CMASK comparison. It is used to set both comparisons (greater than and less than) between CMASK and the counter value.

CMASK (Counter mask) field (bits 24 through 31):

The CMASK field value and the number of events generated in one clock cycle are compared with each other. The counters value gets incremented only if the number of events generated is more than the CMASK value else does not increment.

3.4 READING/WRITING COUNTER:

In general, in core architectural level uses instructions like RDMSR, RDPMC and WRMSR for reading and writing to the counters.

3.4.1. RDMSR:

Reads the contents of a 64-bit model specific register (MSR) specified in the ECX register into registers EDX:EAX. (On processors that support the Intel 64 architecture, the high-order 32 bits of RCX are ignored.) The EDX register is loaded with the high-order 32 bits of the MSR and the EAX register is loaded with the low-order 32 bits.

Table 3.1 RDMSR [11].

RDMSR—Read from Model Specific Register

Opcode*	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
0F 32	RDMSR	ZO	Valid	Valid	Read MSR specified by ECX into EDX:EAX.

3.4.2. RDPMC:

The EAX register is loaded with the low-order 32 bits. The EDX register is loaded with the supported high-order bits of the counter. The number of high-order bits loaded into EDX is implementation specific on processors that do no support architectural performance monitoring.

Table 3.2 RDPMC [11].

RDPMC—Read Performance-Monitoring Counters

Opcode*	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
0F 33	RDPMC	ZO	Valid	Valid	Read performance-monitoring counter specified by ECX into EDX:EAX.

3.4.3 WRMSR:

Writes the contents of registers EDX:EAX into the 64-bit model specific register (MSR) specified in the ECX register. (On processors that support the Intel 64 architecture, the high-order 32 bits of RCX are ignored.) The contents of the EDX register are copied to high-order 32 bits of the selected MSR and the contents of the EAX register are copied to low-order 32 bits of the MSR.

Figure 3.3 WRMSR [11].

WRMSR—Write to Model Specific Register

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
0F 30	WRMSR	ZO	Valid	Valid	Write the value in EDX:EAX to MSR specified by ECX.

3.5 INSIDE PERFMON

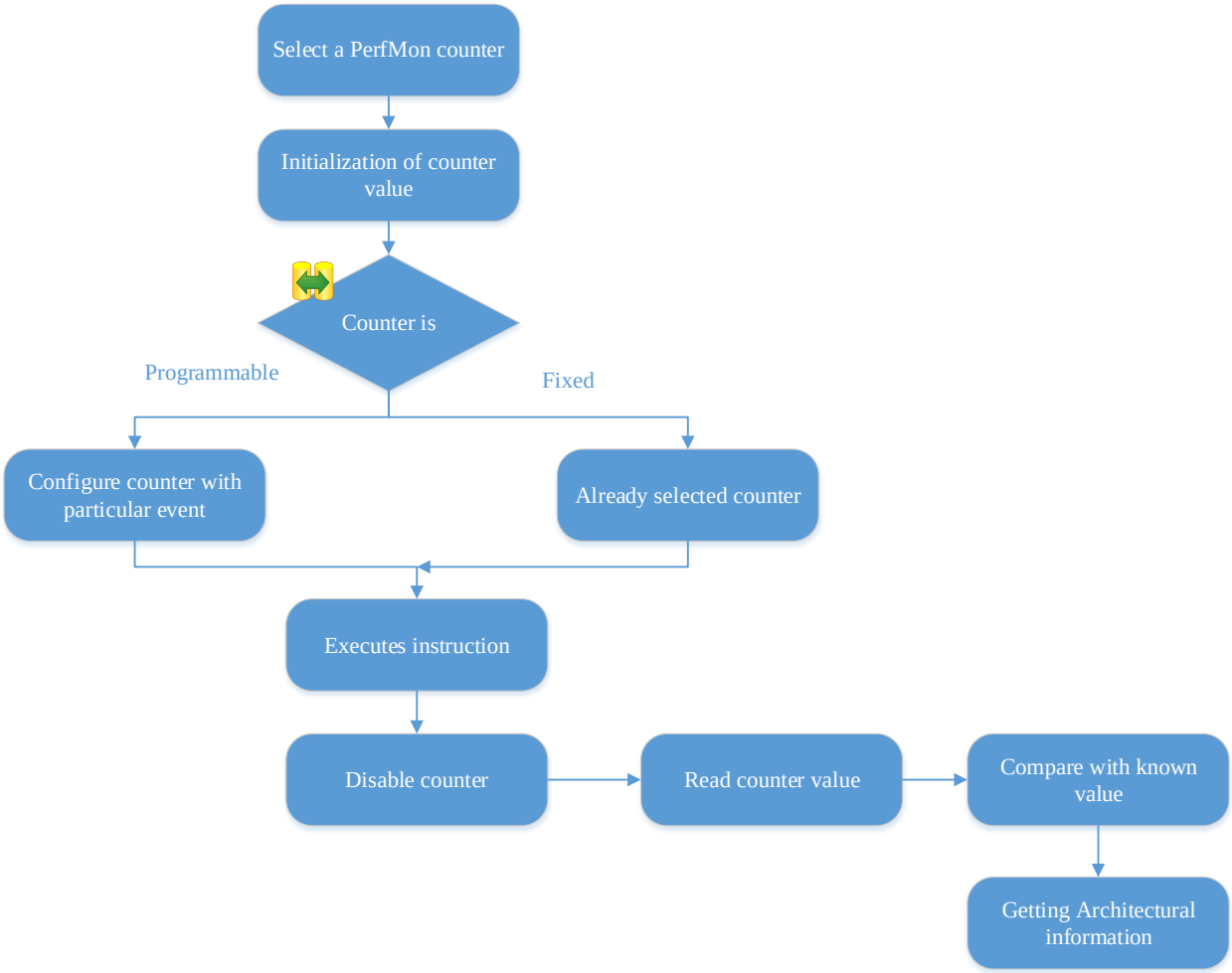


Figure 3.7 Perfmon counter execution flow.

3.6 PERFORMANCE COUNTERS

Performance monitoring counters are unique registers on a CPU used to recognize bottlenecks and other execution blocking attributes of a running project. These counters are utilized to distinguish how regularly certain hardware events, for example, store misses happen amid the execution of a program. With a specific end goal to enhance the execution of an application, first the program is stacked, counters are reset, the program runs and counters are perused. The potential performance enhancement can be determined by a programmer from these values of counters. These changes adjust the code to enhance the utilization of the branch indicator, memory, or some other bit of hardware. Once recognized, the programmer can revise the code and rerun the program to search for more bottlenecks.

As per project definition and requirements the number of Counters will vary. The Performance Monitoring unit contains two types of registers. They are General purpose Counters which is also known as Programmable Counters and second one is Fixed counters. Programmable counters will be configured with particular events to monitor that events status through put the system ON time. For example, the programmable counters will be configured with the events like No. Of Instructions retired, LLC reference, LLC miss, Branch Instruction Retired, Branch Misses Retired etc. Fixed counters are maintaining to monitor only certain events like INST_RETIRED.ANY, CPU_CLK_UNHALTED.THREAD,CPU_CLK_UNHALTED.CORE,CPU_CLK_UNHALTED.REF_TSC.

3.7 HARDWARE COUNTERS IN EXISTING MACHINE

Most present day processors contain hardware performance counters. While these counters are valuable to help recognize issues for a given program on a specific processor, hardware performance counters are processor subordinate. This implies developers can tune their code for a particular processor however not for any self-assertive usage of the ISA. The way Intel executes x86 execution counters is unique in relation to the way AMD does. Indeed, even inside Intel, performance counters have changed after some time; the counters on the P4 are interfaced uniquely in contrast to the ones on the Intel Core Duo. By the by, performance counters have a standard technique for usage that is utilized as a part of all CPUs. Table 2.1 is an example of the no. of counters and event detectors for specific processors.

Table 3.4 Number of counters and events in certain processors [12].

CPU	COUNTERS	EVENTS
Pentium	2	38
Pentium4	18	44
Alpha	3	66
AMD	4	84

A performance counter is an extraordinary enrolls in the CPU that is set up to increase the count value when it gets a particular event signal. As portrayed, the no. of counters is not as much as the number of the events detected. Since performance counters don't specifically enhance execution (or spare power) they are one of the last highlights intended for a processor. Besides, on the grounds that processor region is saved (to cut costs) the number of performance counters is low. By and large, most processors have 4 fixed and 8 general purpose counters, with the uncommon special case of the Pentium 4, which has 18.

Event detectors are utilized to distinguish performance-oriented events, for example, store misses and branch expectations. Table 2.2 depicts normal occasions that occur in current processors.
Intel considered two outlines: one which contained a counter in the usage of every event detector; or another, made out of gatherings of counters spread out all through the whole processor. The last was picked in light of zone preservation which thus cut down on creation costs. To really utilize these counters a developer should first set the CESR for the correct events which must be executed under privilege level 0.

CESR is an acronym for control-and-event-select monitoring register which is used to control incoming event. Subsequently, the bit should first set up the counters to collect the information. To do this, the portion must write to the CESR utilizing the wrmsr command.
Here is a case of setting up the Intel processor to utilize performance counters:

```
wrmsr 0x11 , 0 x00690063        ; initialize the CESR (0 x11 ) to have :
                                ; count e r 0 count number of read misses
                                ; to the data cache , and
                                ; counter 1 count number of memory accesses
                                ; To the data cache.
                                ; Both ar e s e t to always read no matter the
                                ; Privileged level.
wrmsr 0x12 , 0x0                ; Cl ear s count e r 0(0 x12 )
wrmsr 0x13 , 0x0                ; Cl ear s count e r 1(0 x13 )
```

Now, in user space, the programmer must read these counters before and after the code is profiled.

```
mov ecx , 0                     ; select counter 0

rdpmc                           ; read counter specified by eac into register eax
mov ebx , eax                   ; save count e r into register ebx
. . .                           ; the profiled code executes
```

Table 3.5 Common detectable events in processors [11-12].

S no.	Name of the Event	Description
1.	Number of cycles	Keeps track of the number occurred cycles
2.	Number of CPU loads	The number of CPU loads
3.	Data Cache accesses	Number of accesses to the L1 Data cache
4	Data Cache misses	Number of data cache misses. Usually you can configure to only count when in a certain Cache Coherency state (MOESI) and/or reads/writes.
5	Data cache refills	Number of times the Data cache has been refilled by the L2 cache or System memory. Usually this counter is configurable to count only refills from either L2 or System and the state of the cache line was in (MOESI).
6	L2 Cache accesses	Number of accesses to the L1 Data cache
7	L2 Cache misses	Number of data cache misses. Usually you can configure to only count when in a certain coherency state.
8	L2 Cache refills	Number of times the Data cache has been refilled by the L2 cache or System memory. Usually this counter is configurable to count only refills from either L2 or System and the state of the cache line was in (MOESI).
9	Instruction Cache fetches	The number of instruction fetch for the instruction cache.
10	Instruction Cache misses	Number of misses to the instruction cache. Usually this can be broken up into misses that hit the L2 cache or must go to System Memory.
11	Instruction Cache stall	Number of cycles the fetcher stalled. Usually this happens from branch misses or cache misses.
12	Retired instructions	Number of retired, or committed, instructions.
13	Retired uops	Number of retired, or committed, μops
14	Retired branches	Number of retired, or committed, branches
15	Retired mispredict branches	Number of retired, or committed, Mispredicted branches
16	Retired mispredict taken branches	Number of retired, or committed, Mispredicted taken Branches
17	Interrupts	Number of Interrupts.
18	Dispatched stalls	Number of stalls generated. Generally this event can be configured for different reasons such as full reservation stations or aborted branches.

```
mov ecx , 0                ; select cunter0
rdpmc                      ; read counter specified by eac into register eax
sub eax , ebx              ; write the total number of events into register eax
                           ; This assumes that register ebx was not touched.
                           ; You may have to save ebx to memory.
```

Presently enroll eax has the number of read information store misses that happened in the profiled code. On the off chance that the coder additionally utilized counter 1 for add up to reserve gets to, the coder can derive the normal store miss rate. In the event that the miss rate is huge then the programmer can adjust the profiled code to take more care of information reserve misses, through better store arrangement or general memory utilize. Once the addresses are set up, the coder runs the profiled code again to check whether any progressions happen.

CHAPTER 4

PCM (Performance Counter Monitor)

4.1 INTRODUCTION

In the most recent decades complexity of computing systems has enormously increased. Out-of-order execution, simultaneous multithreading, non-uniform memory and hierarchical cache subsystem have a massive impact on the performance and compute capacity of modern processors.

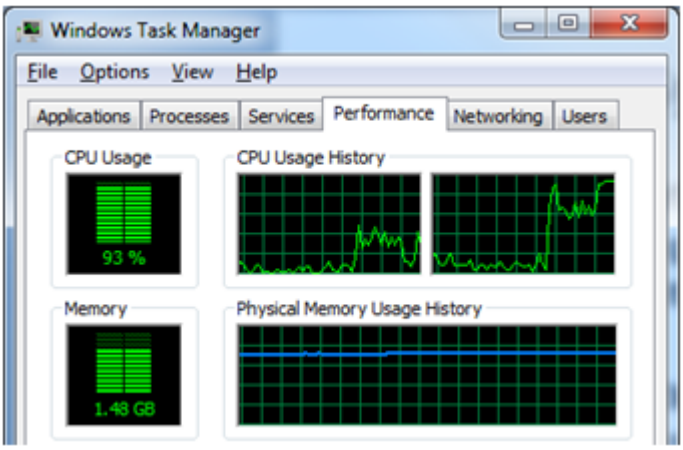


Figure 4.1 CPU Utilization [22].

The software which can understands and flexibly adjusts to resource utilization of modern processors has performance and power advantages. The Intel® Performance Counter Monitor gives test C++ schedules and utilities to appraise the inward asset usage of the most recent Intel® Xeon® and Core™ processors and pick up a noteworthy execution and increase in performance .

4.2 UTILIZATION OF CPU

CPU usage number got from working framework (OS) is a metric that has been utilized for some, reasons like product sizing, compute capacity, job scheduling, et cetera. The present usage of this metric demonstrates the part of slots that the CPU scheduler in the OS could relegate to execution of running projects or the OS itself; whatever remains of the time is sit without moving. For figure bound workloads, the CPU use metric ascertained along these lines anticipated the rest of the CPU limit extremely well for designs of 80ies that had considerably more uniform and unsurprising execution contrasted with present day frameworks. The advances in PC design made this calculation a problematic metric due to presentation of multi core and multi CPU frameworks, multi-level stores, non-uniform memory, concurrent multithreading (SMT), pipelining, out-of-order execution, and so forth [22].

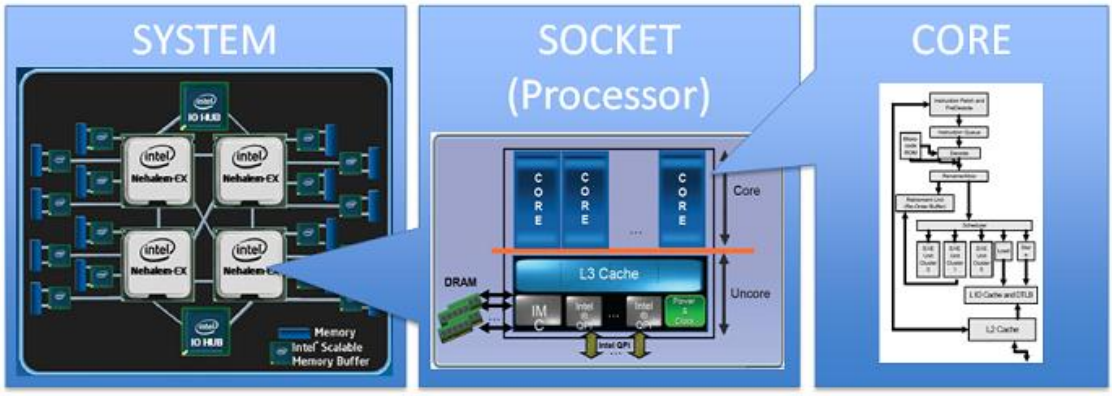


Figure 4.2 The complexity of a modern multi-processor, multi-core system [22].

A prominent example is the non-linear CPU utilization on processors with Intel® Hyper-Threading Technology (Intel® HT Technology). Intel® HT technology is a great performance feature that can boost performance by up to 30%. However, HT-unaware end users get easily confused by the reported CPU utilization: Consider an application that runs a single thread on each physical core. Then, the reported CPU utilization is 50% even though the application can use up to 70%-100% of the execution units.

A different example is the CPU utilization for "memory throughput"-intensive workloads on multi-core systems. The bandwidth test "stream" already saturates the capacity of memory controller with fewer threads than there are cores available.

4.3 ABSTRACTION LEVELS OF PMU

The good news is that Intel processors already provide the capability to monitor performance events inside processors. In order to obtain a more precise picture of CPU resource utilization will rely on the dynamic data obtained from the so-called performance monitoring units (PMU) implemented in Intel's processors. Concentrate on the advanced feature set available in the current Intel® Xeon® 5500, 5600, 7500, E5, E7 and Core i7 processor series [26].

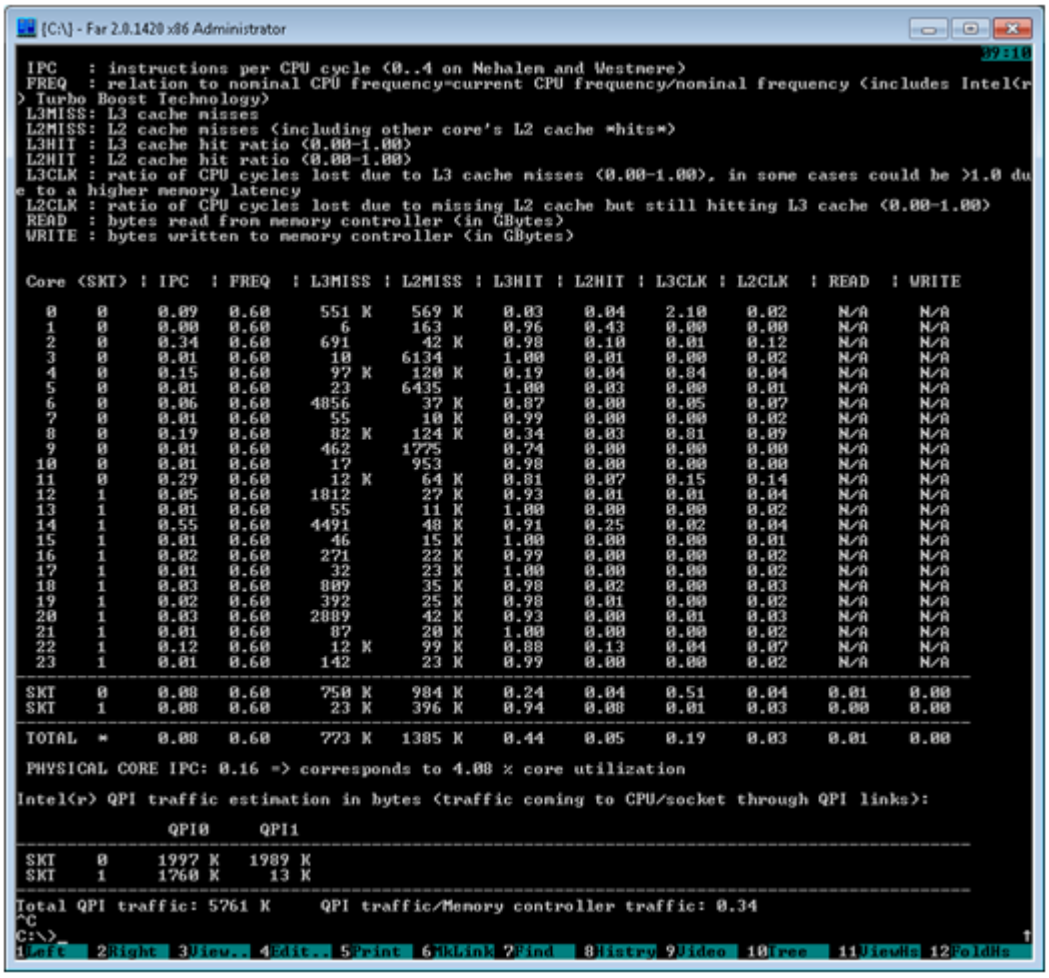


Figure 4.3 PCM command line version [22].

One have implemented a basic set of routines with a high level interface that are callable from user C++ application and provide various CPU performance metrics in real-time. In contrast to other existing frameworks like PAPI and Linux "perf" support not only core but also uncore PMUs of Intel processors (including the recent Intel® Xeon® E7 processor series) [22]. The uncore is the part of the processor that contains the integrated memory controller and the Intel® QuickPath Interconnect to the other processors and the I/O hub. In total, the following metrics are supported:

- Core: instructions retired, elapsed core clock ticks, core frequency including Intel® Turbo boost technology, L2 cache hits and misses, L3 cache misses and hits (including or excluding snoops).
- Uncore: read bytes from memory controller(s), bytes written to memory controller(s), data traffic transferred by the Intel® QuickPath Interconnect links.

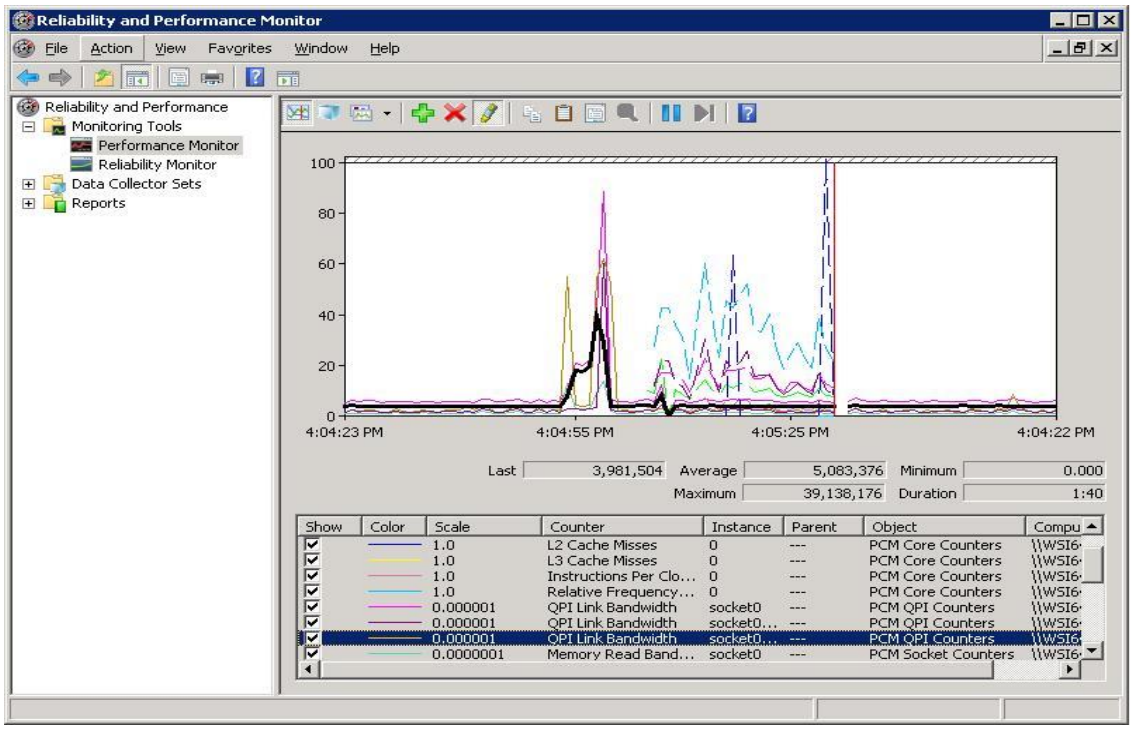


Figure 4.4 Perfmon showing data from Intel® PCM v1.7 [22].

4.4 PCM INSIDE YOUR PROGRAM

Now a day it has become very easy to monitor the processor metrics in your application, library provides the abstraction layer. The performance counters needed to be initialized before they are being used. The code is run and the counter state is collected or captured before and after the code run. Distinct routines capture the counter value for cores, socket and the full system. After that it will store the state in the corresponding data structure. Below code snippet is an example of a usage of a counters.

```
PCM * m = PCM::getInstance();

// initializing the program counters, if failure comes than just exit

if (m->program() != PCM::Success) return;

State of the system counter before_sstate = getSystemCounterState();

[code run here]

State of the System Counter after_sstate = getSystemCounterState();

count<< "Instructions per clock:" << getIPC(before_sstate,after_sstate)

<< "L3 cache hit ratio:" << getL3CacheHitRatio(before_sstate,after_sstate)

<< "Bytes read:" << getBytesReadFromMC(before_sstate,after_sstate)
```

CHAPTER 5

RESULTS AND DISCUSSION

5.1 INTRODUCTION

Regressions reports and results are discussed here. Toolchain is updated for the next generation model. Running the various regressions and checking the results or pass rate of all the performance events. Checking the performance event count for both the model RTL as well as c-based reference model. If count diff is not under some defined threshold value than need to debug and find out the root cause. Total pass rate of various regressions in different work week and pass rate of each event after enhancement in the toolchain is discussed.

5.2 REGRESSION

Regression suggests running a heap of tests that continues running on DUT to uncover performance bugs. It is commonly done in seven days by week way. The tests are continue running with self-assertive seeds which give the chance to find corner case bugs. Pass rate got demonstrates the idea of plan. The tests are made to consolidate Code scope, Functional extension, Toggle scope, Frequency scope. The tests can be changed or new tests can be added to affirm the DUT in different conditions. The tests which are failed are arranged into "buckets" for investigating which should be fixed by the proprietors. The performance bugs obtained can be RTL bug or bug in checker or detail bugs which are taken after and settled. Pursuing week regression is helpful to change the wrong fixes of bugs and track the idea of plan with the fixes made with each watched bug. Pass rate decides the dependability of the model and empowers the element proprietor to check if the model is steady or not. By running irregular seeds different situations are secured every week with the goal that the dependability is enhanced step by step. Gathering all the comparative disappointments into a solitary gathering troubleshoots the things quicker.

5.3 SIMULATION

Simulation is a procedure of checking of an activity of a genuine function for the given time. It exhibits whether the correct yield is acquired in view of the information given as info. There are a few sorts of simulation that can be performed like hardware and software simulation. There are a few segments of recreation like checking the assertion, cover bugs, RTL bugs and so forth. These can be because of numerous components like switch being incapacitated, fault in design, checker disappointment and so on. Such can be gotten over some undefined time frame by running the regression. Reproduction disappointments are principally because of hang blunders or memory issue. At the point when an intruder comes, the subroutine gets executed the directions gets hanged at some place and can't return. Such blunders cause parcel of disappointments in simulation.

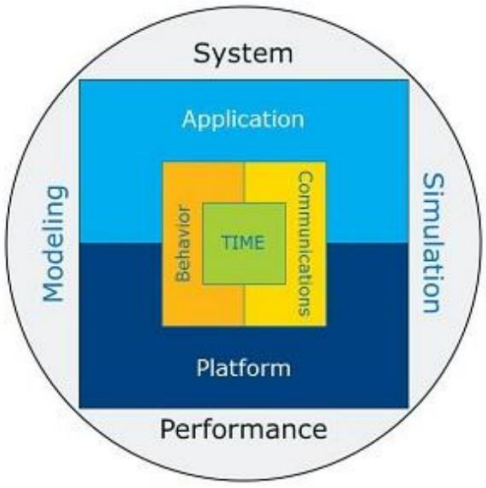


Figure 5.1 Simulation Framework for Intel Studio[12].

5.4 EMULATION

It is speedier than simulation in which the correct replica hardware is displayed in programming for validation. For instance, a child playing bicycle hustling diversion in computer game is a recreation however when he rides a displayed bicycle continuously in a shopping center is a copying. Emulation is a genuine method for testing the correct hardware. At the point when the chip tests itself utilizing the inbuilt vectors accessible then it is named as in-circuit emulation. Testing in hardware by interfacing all parts includes colossal manual exertion thus imitating replaces those hardware by programming. It shows the real time behavior.

5.5 REGRESSION REPORTS AND RESULTS

Running regressions for different projects to check the various snippets lists test. This will show us the number of test passing or failing in the project. The results of the regressions get logged in the main database. Based on the results will check the passing or failing performance events.

ID	Task Name	Status	WL	WR	Run	Succ	Fail	Skip	Progress	Started	Finished	Cost
12442	[redacted]monikach_2018-05-10_SMT_core-[redacted]18ww18d_TopdownL1_Relationship_May10_14_33_26	Completed	0	0	0	397	65	0		05/10/2018 14:33:56	05/11/2018 12:	$107.11
12436	[redacted]onikach_2018-05-10_SMT_core-[redacted]18ww15b_TopdownL1_Relationship_May10_14_32_54	Running	5	0	2	401	54	0		05/10/2018 14:33:55		$112.75
12415	[redacted]monikach_2018-05-10_Base_core-[redacted]8ww18e_May10_10_21_41	Completed	0	0	0	800	103	0		05/10/2018 10:21:49	05/11/2018 03:	$162.39
12307	[redacted]monikach_2018-05-09_SMT_core-[redacted]18ww18d_TopdownL1_Relationship_May09_11_55_04	Completed	0	0	0	211	251	0		05/09/2018 11:55:07	05/10/2018 00:	$71.71
12301	[redacted]onikach_2018-05-09_SMT_core-[redacted]18ww15b_TopdownL1_Relationship_May09_11_54_46	Running	5	0	1	208	248	0		05/09/2018 11:54:55		$75.49
12289	[redacted]monikach_2018-05-09_SMT_core-[redacted]18ww18e_May09_08_59_14	Completed	0	0	0	347	115	0		05/09/2018 08:59:20	05/10/2018 08:	$111.01
12337	[redacted]monikach_2018-05-09_Base_core-[redacted]18ww18d_TopdownL1_Relationship_May09_14_32_05	Completed	0	0	0	872	31	0		05/09/2018 14:32:08	05/10/2018 03:	$146.59
12331	[redacted]onikach_2018-05-09_Base_core-[redacted]18ww15b_TopdownL1_Relationship_May09_14_31_33	Running	5	0	3	862	33	0		05/09/2018 14:31:38		$161.40
12295	[redacted]monikach_2018-05-09_[redacted]throughput_nomem_core-glc-a0-18ww18e_May09_09_55_05	Completed	0	0	0	7	0	0		05/09/2018 09:55:10	05/09/2018 12:	$0.12
12265	[redacted]monikach_2018-05-08_Base_core-[redacted]18ww18d_TopdownL1_Relationship_May08_11_54_36	Completed	0	0	0	874	29	0		05/08/2018 11:54:40	05/08/2018 23:	$145.75
12259	[redacted]onikach_2018-05-08_Base_core-[redacted]18ww15b_TopdownL1_Relationship_May08_11_53_57	Running	5	0	1	872	25	0		05/08/2018 11:54:11		$154.58
12277	[redacted]monikach_2018-05-08_[redacted]throughput_nomem_core-glc-a0-18ww18e_May08_13_38_49	Completed	0	0	0	6	1	0		05/08/2018 13:38:55	05/08/2018 16:	$0.15
	Total tasks: 12		20	0	7	5,857	955	0				$1,249.06

Figure 5.2 Regression results for project1.

It will depend upon the count value of the performance event. If the count value of the performance events is varying between the RTL model and the simulator that means it will affect the performance of the system.

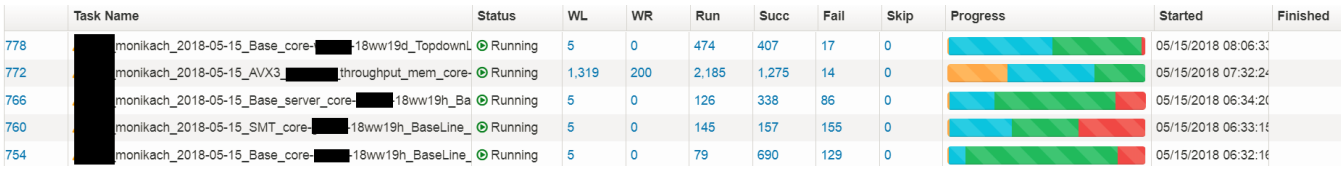


Figure 5.3 Regression results for project2.

Below is the table which shows the pass rate of the test runs in different week.

Table 5.1 Pass rate of regressions in different ww.

Work week	**Total test**	**Passing test**	**Failing test**	**Pass rate**
Ww10	462	397	65	85.9%
Ww11	903	874	29	96%
Ww12	895	862	33	96.3%
Ww13	903	872	31	96.5%
Ww14	897	872	25	97%

5.6 TOOLS CHANGES TO ENABLE THE NEW PERFMONEVENTS

The main work is done to update the toolchain for the next generation project. Enabling the perfmon events for the new project and updating the toolchain to support them. Various runs and regressions are made to enable the perfmon events in the latest project. And debugged the toolchain issues which were coming further after enabling them.

UPDATION

Added new Perfmetrics events in the latest model or project

Some new Perfmetrics events are added in the latest project and updation is done in the toolchain to support these new events. So, the new added events can count the performance value for both RTL and simulator. And also, they should dump the perfmon metrics event count value to the RTL and other log files.

Problems faced

The new perf events were enabled in the model but there was no RTL counts because of this most of test cases were facing the Max Diff issues.

Solution: - After debugging the failures it comes into account that the modeled events name in both RTL and reference simulator were not the same. So, fix is done in the main toolchain and subsequent scripts to ensure that each and every event has the correct name and should modeled correctly in both.

After enabling the new events in the latest model and all the update in the toolchain, checking the pass rate of the performance events in the main database and selecting the failing performance events and compare the event count in RTL and c-based reference model. If the events count difference is more than the threshold value specified for each performance event than need to debug the failure. There are more than 200 performance events enabled in the next project. The below table shows the pass rate of some event after debugging the failures in the latest project. The pass rate can be increased by tracking the performance event.

Table 5.2 Pass rate of events in different project.

Event name	**Status**	**Pass rate Project 1**	**Passrate Project 2**
Number of cycle	Passing	**0.99**	**0.97**
Number of CPU load	Passing	**0.99**	**0.98**
Data cache accesses	Passing	**0.99**	**0.99**
Data cache misses	Passing	**0.86**	**0.81**
Data cache refills	Missing coverage	**0.00**	**0.75**
L2 cache accesses	Passing	**0.98**	**0.98**
L2 cache misses	Passing	**1.00**	**1.00**
L2 cache refills	RTL_bug	**0.90**	**1.00**
Instruction cache fetches	Passing	**0.98**	**0.96**
Instruction cache misses	Passing	**0.98**	**0.97**
Instruction fetch stall	Passing	**0.97**	**0.94**
Retired instructions	Toolchain	**0.89**	**0.80**
Retired uops	Passing	**0.99**	**0.95**
Retired branches	Passing	**0.99**	**0.94**
Retired Mispredict branches	Missing coverage	**0.00**	**0.93**
Retired Mispredict branches taken	Passing	**0.98**	**0.98**
L3 cache hit	Passing	**0.98**	**0.95**
Dispatched stalls	Passing	**0.96**	**0.94**
Cycle activity stalls l1 miss	RTL_bug	**0.80**	**0.75**
Cycle activity stalls l2 miss	Passing	**0.94**	**0.91**
Cycle activity stalls l3 miss	Missing coverage	**0.00**	**0.94**
Cycle activity stalls mem	Passing	**0.98**	**1.00**
Mem instruction retired	Passing	**0.99**	**1.00**

CHAPTER6

CONCLUSION AND FUTURE SCOPE

6.1 INTRODUCTION

This chapter concludes the pre-silicon validation of performance monitoring counters in the core level also the future scope of the work. The PMU unit which is built inside out of order unit in a processor used to determine the performance parameters. And further enhancement to toolchain for checking new events.

6.2 CONCLUSION

The general performance validation is based upon the understanding of micro architecture design. And the performance of an application can be evaluated by using various perf events. Now a days industries are moving to multicore architecture, therefore more performance events must be added to evaluate the performance of the system. That will help to find the bottlenecks in the multiprocessor application. The instruction execution in the super pipeline chooses the performance count. A few operators are quite certain for the execution approval as execution related test generators, checkers and trackers to track some performance related events. So an ever increasing number of situations can be checked before sending to the back end group. Numerous examines are being done to both increment the battery limit and furthermore to build the performance of the processor.

The basic idea is to keep on running the various snippets lists which will keep track on the each and every performance events. Running them will create the enough results and will check the performance events are getting the correct results or not. Enabling the new feature for the performance events to have more and more enhancement in the architecture and can improve performance.

The main work is done to enable the new perf events in the latest model and all the subsequent changes done to support them in the toolchain. Further check the performance events for a c- based reference model that will tests on both model i.e. c- based simulator and RTL model. After numerous examinations the comparison studies are executed between the models. Various and distinct configurations runs are made to check different perf events. Which will ensure that RTL and Behavioral model should match and the performance event count should match. This thesis proposed tools enhancement for checking new performance events in the core.

6.3 FUTURE SCOPE

Now a day's advancement in technology is increasing every day. The performance monitoring unit is used to determine the performance of a system. All the Intel processors provide the capability to monitor performance events inside processors. Future work will involve a more detailed study of this unit to enhance performance of an application or system. From this work multicore applications can be evaluated to find performance improving characteristics. Various

perfmetrics events are checked by running different configuration. Further enhancement to the toolchain to support the performance counters in the next generation core model.

REFERENCES

[1] Korn, W., Teller, P.J. and Castillo, G., 2001, April. Just how accurate are performance counters? *IEEE International Conference on Performance, Computing, and Communications, 2001*. (pp. 303-310).

[2] Sprunt, B., 2002. The basics of performance-monitoring hardware. *IEEE Micro*, *22*(4), pp.64-71.

[3] Maxwell, M., Teller, P., Salayandia, L. and Moore, S., 2002, October. Accuracy of performance monitoring hardware. In *Proceedings of the Los Alamos Computer Science Institute Symposium (LACSI'02)*.

[4] Eranian, S., 2006, July. Perfmon2: a flexible performance monitoring interface for Linux. In *Proc. of the 2006 Ottawa Linux Symposium* (pp. 269-288).

[5] Eyerman, S., Eeckhout, L., Karkhanis, T. and Smith, J.E., 2007. A top-down approach to architecting CPI component performance counters. *IEEE micro*, *27*(1).

[6] Weaver, V.M. and McKee, S.A., 2008, September. Can hardware performance counters be trusted? *IISWC 2008. IEEE International Symposium on Workload Characterization, 2008* (pp. 141-150).

[7] Diamond, J., McCalpin, J.D., Burtscher, M., Kim, B.D., Keckler, S.W. and Browne, J.C., 2010. Making sense of performance counter measurements on supercomputing applications. *Technical Report TR-10-25*.

[8] Weaver, V.M., Terpstra, D. and Moore, S., 2013, April. Non-determinism and overcount on modern hardware performance counter implementations. *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, (pp. 215-224).

[9] Yasin, A., 2014, March. A top-down method for performance analysis and counters architecture. *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), 2014* (pp. 35-44).

[10] Saez, J.C., Pousa, A., Rodriíguez-Rodriíguez, R., Castro, F. and Prieto-Matias, M., 2017. PMCTrack: delivering performance monitoring counter support to the OS scheduler. *The Computer Journal*, *60*(1), pp.60-85.

[11] Pentium III Intel Architecture Software Developer's Manual, *[online] Available:*http://www.intel.com/design/PentiumIII/manuals/

[12] Intelpedia[Intelpedia (n.d.)]

[13] Desikan, R., Burger, D. and Keckler, S.W., 2001, June. Measuring experimental error in microprocessor simulation. In *Proceedings of the 28th annual international symposium on Computer architecture* (pp. 266-277).

[14] Zagha, M., Larson, B., Turner, S. and Itzkowitz, M., 1996, November. Performance analysis using the MIPS R10000 performance counters. In *Proceedings of the 1996 ACM/IEEE conference on Supercomputing* (p. 16).

[15] Palacharla, S., Jouppi, N.P. and Smith, J.E., 1997. *Complexity-effective superscalar processors* (Vol. 25, No. 2, pp. 206-218).

[16] Performance application programming interface. http://icl.cs.utk.edu/papi/, 2008.

[17] Williams, S., Waterman, A. and Patterson, D., 2009. Roofline: an insightful visual performance model for multicore architectures. *Communications of the ACM*, *52*(4), pp.65-76.

[18]"Papi: Performance application programming interface." [Online]. Available: http://icl.cs.utk.edu/papi

[19] Dean, J., Hicks, J.E., Waldspurger, C.A., Weihl, W.E. and Chrysos, G., 1997, December. ProfileMe: Hardware support for instruction-level profiling on out-of-order processors. In *Proceedings of the 30th annual ACM/IEEE international symposium on Microarchitecture* (pp. 292-302).

[20] Araiza, R., Aguilera, M.G., Pham, T. and Teller, P.J., 2005, October. Towards a cross-platform microbenchmark suite for evaluating hardware performance counter data. In *Proceedings of the 2005 conference on Diversity in computing*(pp. 36-39).

[21] Etsion, Yoav Computer Architecture. Available at

https://iis-people.ee.ethz.ch/~gmichi/asocd/addinfo/Out-of-Order_execution.pdf/ (Accessed on 9th January)

[22] Performance Monitoring counter. Available at https://software.intel.com/en-us/articles/intelperformance-counter-monitor (Accesed on 21st February)

[23] Saez, J.C., Fedorova, A., Koufaty, D. and Prieto, M., 2012. Leveraging core specialization via OS scheduling to improve performance on asymmetric multicore systems. *ACM Transactions on Computer Systems (TOCS)*, *30*(2), p.6.

[24] Van Craeynest, K., Akram, S., Heirman, W., Jaleel, A. and Eeckhout, L., 2013, September. Fairness-aware scheduling on single-ISA heterogeneous multi-cores. In *2013 22nd*

International Conference on Parallel Architectures and Compilation Techniques (PACT), (pp. 177-187).

[25] Perf (2015). Perf wiki tutorial on perf. https://perf. wiki.kernel.org/index.php. Accessed: 201501-20.

[26] Intel, "Intel 64 and IA-32 Architectures Optimization Reference Manual, Section 3.4.2.4: Optimizing the Loop Stream Detector," http://www.intel.com/products/processor/manuals/.

[27] T. Austin. Simplescalar 4.0 release note. Available at http://www.simplescalar.com/. (Accessed on 6th march)

[28] Bellard, F., 2005, April. QEMU, a fast and portable dynamic translator. In *USENIX Annual Technical Conference, FREENIX Track* (Vol. 41, p. 46)

[29] Black, B., Huang, A.S., Lipasti, M.H. and Shen, J.P., 1996, October. Can trace-driven simulators accurately predict superscalar performance? *IEEE International Conference on Computer Design: VLSI in Computers and Processors* (pp. 478-485).

[30] Contreras, G., Martonosi, M., Peng, J., Ju, R. and Lueh, G.Y., 2004, June. XTREM: a power simulator for the Intel XScale® core. In *ACM Sigplan Notices* (Vol. 39, No. 7, pp. 115-125).

[31] DeRose, L.A., 2001, August. The hardware performance monitor toolkit. In *European Conference on Parallel Processing* (pp. 122-132).

601662011_Monika.pdf

ORIGINALITY REPORT

9%	%	9%	%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	"VLSI Physical Design Automation", Algorithms for VLSI Physical Design Automation, 2002 Publication	1%
2	J. C. Saez, A. Pousa, R. Rodriíguez-Rodriíguez, F. Castro, M. Prieto-Matias. "PMCTrack: Delivering Performance Monitoring Counter Support to the OS Scheduler", The Computer Journal, 2017 Publication	1%
3	Vincent M. Weaver. "Can hardware performance counters be trusted?", 2008 IEEE International Symposium on Workload Characterization, 10/2008 Publication	1%
4	Ahmad Yasin. "A Top-Down method for performance analysis and counters architecture", 2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), 2014 Publication	1%

Monika