A systematic approach in driving the model for optimization and efficiency improvement in Ingredients and Platform Validation for Intel Client Platforms(2012-13)

Prepared By

Manthan V. Shah

10MCEC16

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

NIRMA UNIVERSITY

AHMEDABAD-382481

May 2012

A systematic approach in driving the model for optimization and efficiency improvement in Ingredients and Platform Validation for Intel Client Platforms(2012-13)

Major Project

Submitted in partial fulfillment of the requirements

For the degree of

Master of Technology in Computer Science and Engineering

Prepared By

Manthan V. Shah

10MCEC16

GUIDED BY :

Mukesh Kothari

S Ravishankar

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

NIRMA UNIVERSITY

AHMEDABAD-382481

May 2012

DECLARATION

I, **Manthan Shah**, **10MCEC16**, give undertaking that the Major Project entitled **"A systematic approach in driving the model for optimization and efficiency improvement in Ingredients and Platform Validation for Intel Client Platforms(2012-13)"** submitted by me, towards the partial fulfillment of the requirements for the degree of Master of Technology in Institute of Technology of Nirma University, Ahmedabad, is the original work carried out by me and I give assurance that no attempt of plagiarism has been made. I understand that in the event of any similarity found subsequently with any published work or any dissertation work elsewhere; it will result in severe disciplinary action.

Manthan Shah

Certificate

This is to certify that the Major Project entitled "**A systematic approach in driving the model for optimization and efficiency improvement in Ingredients and Platform Validation for Intel Client Platforms(2012-13)**" submitted by **Manthan V. Shah (10MCEC16)**,towards the partial fulfillment of the requirements for the degree of Master of Technology in Computer Science and Engineering of Nirma University , Ahmedabad is the record of work carried out by him under my supervision and guidance. In my 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 my knowledge, haven't been submitted to any other university or institution for award of any degree or diploma.

Mukesh Kothari
Project Manager,
Intel Technology India Pvt. Ltd.
Bangalore

Prof. Tejal Upadhyay
Internal Guide,
Institute of Technology,
Nirma University, Ahmedabad

S Ravishankar
Project Guide,
Intel Technology India Pvt. Ltd.
Bangalore

Prof. S. N. Pradhan
Professor,PG Coordinator,
Computer Science and Engineering Department,
Nirma University, Ahmedabad

Dr. Ketan Kotecha
Director,
Institute of Technology,
Nirma University, Ahmedabad

Prof. D. J Patel
Professor,HOD,
Computer Science and Engineering Department,
Nirma University, Ahmedabad

Abstract

Today, the complexity of the computer has grown, with processors and chipsets incorporating millions of the transistors and compatible with dozens of operating system, hundreds of platform components and thousands of hardware devices and software applications. Thus complexity of a platform has created an astronomical number of possible test cases for a platform level validation Process. The coverage includes interoperability with many devices, operating system and components thus raising the number of the test cases to near infinity. Compatibility is also a major issue with all components as platform must be compatible with thousands of third party components.

Clearly, there is no way to test each and every possible combination. Validation test plan must include only set of test cases that are important and can be covered during specified time line because Time to market is crucial factor for any product. Also Test plan must be complete enough to achieve better quality product as the aim of the ingredient and platform validation is to achieve Quality, Compatibility and Reliability across all PC platforms.

In this thesis report, our proposal is to define a systematic approach to achieve optimization and efficiency improvement by considering overall validation process , Test content optimization and Test content Automation for Ingredients and Platform validation (for Intel client platforms) so as to achieve better Quality end products.

Acknowledgements

I am deeply indebted to my thesis supervisor **S, Ravishankar** for his constant guidance and motivation. He has devoted significant amount of his valuable time to plan and discuss the thesis work. Without his experience and insights, it would have been very difficult to do quality work.

I would also like to extend my gratitude to my project manager **Mukesh Kothari** for his constant support.Without his support and motivation , it would have been very difficult to achieve quality results.

I would also like to thank **Prof D. J. Patel** , Head of Department Computer Science , **DR. Ketan Kotecha** , Director of Nirma University ,**Prof S. N. Pradhan** , PG Coordinator Computer Science and also my college Internal Guide **Prof. Tejal Upadhyay** for their constant support and allowing me to do internship in such a big and reputed Organization.

Last, but not the least, no words are enough to acknowledge constant support and sacrifices of my family members because of whom I am able to complete the degree program successfully.

- Manthan V. Shah

10MCEC16

Contents

List of Figures

Chapter 1

Introduction to Post Silicon Validation

Post-silicon validation[1] is the last step in the development of a semiconductor integrated circuit. During the pre-silicon process, engineers test devices in a virtual environment with sophisticated simulation, emulation, and formal verification tools. In contrast, post-silicon validation tests occur on actual devices running at-speed in commercial, real-world system boards using logic analyzer and assertion-based tools.

Large semiconductor companies spend millions creating new components; these are the "sunk costs" of design implementation. Consequently, it is imperative that the new chip function in full and perfect compliance to its specification, and be delivered to the market within tight consumer windows. Even a delay of a few weeks can cost tens of millions of dollars. Post-silicon validation is therefore one of the most highly leveraged steps in successful design implementation.

Compatibility and reliability issues are found and resolved early, and the results are used to further refine Intel's design and manufacturing tools and processes. As a result, the quality and reliability of Intel platforms and components have improved

steadily, even as performance and complexity have continued to climb. There are basically five stages in Intel's comprehensive post silicon validation program:

- Stage 1: System Validation
- Stage 2: Analog Validation
- Stage 3: Compatibility Validation
- Stage 4: Software Validation
- Stage 5: Product Qualification

Stage 1: System Validation

System validation puts the actual component through a comprehensive suite of tests in a real platform environment. The test suites are applied to the Intel CPU, chipset and graphics subsystem, frequency and temperature conditions are intensified to test performance at the extreme corners of the component's specifications.

CPUs

System validation stresses both the architectural and micro-architectural features of the processor, with a focus on cache coherency and multiprocessor environments. Both systematic and random tests are used to cover very deep data space and intensive floating-point demands. System validation tests for the Intel Pentium 4 processor offer a good example of the scope and intensity of the process:

- 2,450 CPU feature tests; 2,000 ancestral architectural tests
- Random instruction testing, 1 trillion instructions per week

- Focused tests (I/O stress testing, millions of chipset feature permutations)
- Manipulate any piece of memory by any CPU in a multiprocessor environment

Chipsets

All chipset features are tested using custom-built system validation boards and test cards running custom software to stress each interface of the chipset. Performance parameters are pushed to extreme limits on all cards and busses concurrently, to validate performance limits and to verify bus compatibility.

Graphics

Specialized tools and test suites are employed for validation testing of the graphics subsystem in all Intel chipsets with integrated graphics. Special test images are created to ensure a rigorous baseline for automated testing. If a test reports even a single wrong bit, the root cause is determined by a validation engineer, so that the problem can be fully resolved to ensure outstanding visual quality with no defects.

Stage 2: Analog Validation

As PC performance demands continue to climb, the electrical integrity of processors and chipsets is vital to ensure reliable operation at high frequencies. Intel's analog validation testing finds failures that can happen in just trillionths of a second. Components are stressed to failure at the extremes of temperature, voltage and frequency. Issues are resolved, and findings are shared with design and production engineers in order to improve Intel's design and production methods.

There are two major aspects to analog validation: Circuit Marginality Validation and Analog Integrity Engineering.

Circuit Marginality Validation (CMV)

During CMV, the test suites that were used during pre-silicon simulation of Intel processors are applied again, but this time with a focus on reliability and performance under extreme operating conditions. Both commercial and custom tests are used to test voltage, frequency and temperature extremes, and to ensure reliable operation within the product's specifications.

Analog Integrity Engineering (AIE)

AIE tests the integrity of the chipset, to make sure the entire platform is electrically robust. Performance is validated under a wide variety of worst-case scenarios. The electrical robustness of the chipset and processor is validated under real world scenarios.

Stage 3: Compatibility Validation

A key advantage of Intel architecture is its wide compatibility with third-party hardware components, software applications and operating systems. Intel components and platforms undergo exhaustive compatibility, stress and concurrency testing with over 20 operating systems, numerous motherboards and add-in cards, more than 150 peripherals, and more than 400 applications. OS testing includes multiple versions of Microsoft Windows, NetWare, SCO, UnixWare and Linux. Application testing includes many of the world's most popular business and multimedia programs, as well as numerous games, industry benchmarks and industry hardware tests. A comprehensive suite of tests is also applied to the component in a heavily networked environment. Massive file transfers and broadcasts test performance and data coherency using a wide range of protocols, including Ethernet, Fast Ethernet, Gigabit Ethernet, and

FibreChannel. In addition to these well-known hardware and software products and protocols, the component is tested with specially designed cards that push test parameters beyond conventional limits, to ensure superior performance under worst-case conditions. For example, multimedia traffic is increased to the bandwidth limit of the PCI bus, to verify that audio and video signals do not break up under peak workloads.

Stage 4: Software Validation

Intel develops all core software components for its PC and server platforms. This includes the BIOS and the drivers that are used for graphics, storage and LAN connectivity. Throughout this process, software validation is tightly integrated with hardware validation to ensure that hardware and software components operate smoothly together. This is essential to validate that the total platform will deliver top performance and reliability in the widest possible range of environments.

Microsoft WHQL Certification testing is performed on all Intel software that is specified for use in a Microsoft Windows operating environment. The WHQL test suite is used in addition to Intel's proprietary tests, to certify Intel drivers and to ensure exceptionally strong validation with Microsoft applications and operating systems. Thanks to Intel's software validation expertise and established processes, WHQL certification throughput has been reduced from weeks to days on most driver releases.

Stage 5: Product Qualification

This is a final phase in Validation cycle, in this stage all the subsystems like graphics, PCI devices are brought together on platform and the behavior of silicon is tested. The compatibility and interoperability of all components on a platform is validated and any customer issues will be resolved. During this final stage, component capabilities are also compared with current end-user expectations. If a product successfully

passes this testing, it is ready to enter the marketplace.

1.1 Problem Statment

Today, the complexity of the computer has grown, with processors and chipsets incorporating millions of the transistors and compatible with dozens of operating system, hundreds of platform components and thousands of hardware devices and software applications. Thus complexity of a platform has created an astronomical number of possible test cases in platform level validation. The coverage includes interoperability and compatibility with many devices, operating system and components thus raising the number of the test cases to near infinity. Clearly, there is no way to test every possible combination.

Test plan must cover all functional areas and also test plan must include limited set of test cases that are important and can be covered in specified time line and with minimum effort. We have to define and prioritize set of test cases that are important and must be included in test plan. We have to define test case based on various factors like different features, customer bus escapes and gaps in various functional areas to achieve full coverage.

So key challenge here is to define and develop effective methodology for complete Platform Validation and finding key defects to ensure the health of the platform.

1.2 Thesis Organization

The rest of the thesis is organized as follows.

Chapter 2, *Literature Survey*, describes the basic architecture of Intel Platform and different platform Ingrediants/Components.

Chapter 3, *BIOS Validation Analysis*, presents the initial problem definition for the BIOS validation and solutions targeting those problems.

In **Chapter 4**, *Platform Stress Analysis*, a new problem statement for platform stressing is identified based on existing stress test content analysis.

Chapter 5, *Background Into Methodology*, describes the new methodology for platform level stressing.

Chapter 6, *Results*, will cover the case studies carried out using presented methodology.

Chapter 7, *Recommendation and Approach*,will cover recommendations for optimizations and improvements in the areas of platform stressing.

Finally, in **Chapter 8** concluding remarks and scope for future work is presented.

Chapter 2

Literature Survey

2.1 Intel Architecture - Platform Overview [2]

The platform is complex with lots of components on it. Every component must work as designed and there shouldn't be any conflicts between the devices on it. The figure below shows the typical diagram of Intel Client platform 2012(Chief River/Maho Bay).
The Chief River platform is a new Tick CPU and PCH hardware architecture, succeeding the Huron River platform. The CPU code name is Ivy Bridge. The PCH code name is Panther Point. The wireless solutions are likely to be Kilmer Peak and Puma Peak. The GbE Ethernet controller is called Lewisville.

The Ivy Bridge CPU is Intel's next generation Tock architecture. IVB features a 22nm Hi-K, 1270 process. It has fully integrated graphics on the same 22nm Hi-K process. The IA core has Intel hyper threading technology, and the next generation turbo boost technology. The Ivy Bridge CPU consists for 5 separate dies spanning the extreme edition, mainstream, and Value solutions. A new segment, consumer ULV also has a dedicated Ivy Bridge Die.

The basis for the mobile platform is the combination of the Ivy Bridge CPU (dual or quad core) with Panther Point PCH. It will also include Lewisville for GbE LAN and 802.3az and Kilmer Peak half minicard for WIFI, WiMax combo. Figure below2.1 shows the block diagram of 2012 platforms. Some of the significant changes from the 2011 Huron River platform include the following.

- Generation 7 graphics support including DX11, DX10.1, DX10, and DX9, with 3DMark06 performance increase of 20
- PCIe Gen3 support on IVB to support high end discrete Gfx cards
- DDR3 memory technology at 1067MHz 1333MHz in a one DIMM per Channel (1DPC). Off roadmap support for 1067 and 1333 exists for 2DPC on quad core processors.
- New Super Speed USB3 and XHCI host controller
- The Legacy PCI bus interface is removed from the mobile PCH.
- USB3 and an XHCI supports 4 dedicated USB3 ports
- 6 Serial ATA (SATA) with 2 FIS based port multiplier support and command-based port multiplier support for all the ports. SATA 6Gbis implemented on two of the non-FIS port.
- 8 PCIE 5Gbports
- Visual quality enhancements - meeting consumer expectations for HD displays
- Wireless Display- for viewing content on TV over wireless from notebook

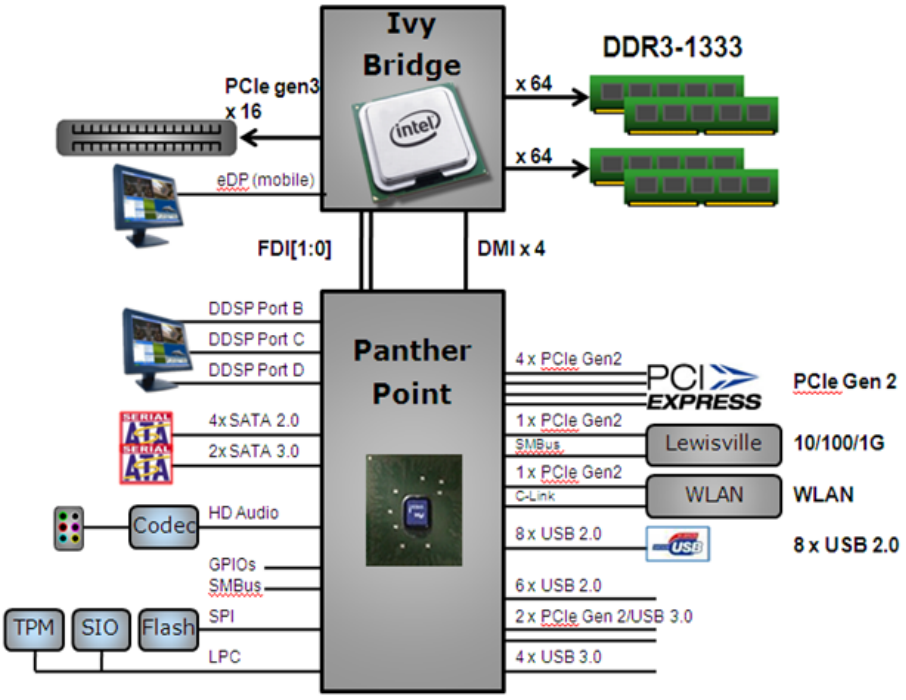


Figure 2.1: High-level block diagram of Chief River / Maho Bay platforms

2.2 Platform Controller Hub (PCH) Architecture Overview

Platform Controller Hub (PCH) is a family of Intel microchips employed in redesigned Intel Hub Architecture chipsets. PCH-based chipsets are designed to address the eventual problem of a bottleneck between the processor and the motherboard. As processing speeds/cores keep increasing, the bandwidth connection between the CPU and the motherboard would reach full capacity and a bottleneck would occur. The speed would be limited by the FSB. As a solution, the new PCH-oriented platform architecture transferred several functions, connections, and controllers belonging to the traditional northbridge and southbridge chipsets and rearranged them between a

new central hub called the PCH and the CPU. In summary, the PCH takes over most of the traditional roles of the southbridge and the few remaining roles traditionally in the northbridge that have not been incorporated into the CPU package.

Before the Platform Controller Hub, a motherboard would have a two piece chipset consisting of an northbridge chip called the MCH and a southbridge chip. The northbridge, later called an memory controller hub (MCH), would have the highest bandwidth functions. The CPU, memory, and AGP or PCI Express graphics slot, if present, would connect to it directly. The northbridge's connection with the CPU, called the front-side bus (FSB), and connection to RAM, called the back-side bus, were each described by data transfer speeds. The southbridge chip, later called an I/O controller hub (ICH), would connect the northbridge to all other lower bandwidth peripherals such as hard drives, CD and floppy drives, Ethernet, keyboard/mouse, PCI cards, system clock, and PCI graphics cards. As CPUs gained more speed and more cores, the connection between the CPU and the northbridge chip would soon be unable to keep up with the CPU thereby slowing the system down. So the connection needed a bigger pipeline.

Several changes have taken place with the evolution to PCH-based chipsets in comparison to the earlier MCH plus ICH based chipsets. The primary change is that the northbridge has been eliminated completely and most of its functions, e.g., the integrated memory controller (IMC) and integrated graphics device (IGD), are now incorporated into the CPU package (often on the same die). Secondly, the PCH now becomes the southbridge and incorporates all of its functions as well as a few of the remaining northbridge functions not subsumed by the CPU (e.g., clocking). Before, the memory RAM and the graphics card would communicate with the northbridge chipset which in turn would communicate with the CPU. Now, memory and graphics card communicate with the CPU within the same package, thereby relieving much of the bandwidth between the processor and the PCH. This means that the PCH is

not connected to the memory or the PCI-Express graphics (PEG) slot. However, the PCH still does have a display controller and a connection to the integrated graphics display if one exists. In addition, the system clock is not a connection any longer and instead fused in with the PCH. Two different connections exist between the PCH and the CPU: Flexible Display Interface (FDI) and Direct Media Interface (DMI). The FDI only exists if an integrated graphics device (IGD) is in the CPU package.

Panther Point PCH is the next generation Platform Controller Hub for the 2012 Platforms and represents the functions that Cougar Point had in the 2011 platform. Panther Point is a minor HW change over Cougar Point, mainly enabling USB 3 support. Following diagram 2.2 shows all the major interfaces supported by the PCH.

Internally, the PCH has following components:

- Power management logic
- System management (TCO reduction) logic
- PC-compatible DMA (8237), Interrupt (8259/APIC), Timer (8245), and Real-Time Clock
- Integrated thermal sensor
- Manageability Engine integrated into Panther Point with Intel iAMT8 support
- Virtualization Engine integrated
- LaGrande Technology support for TPM

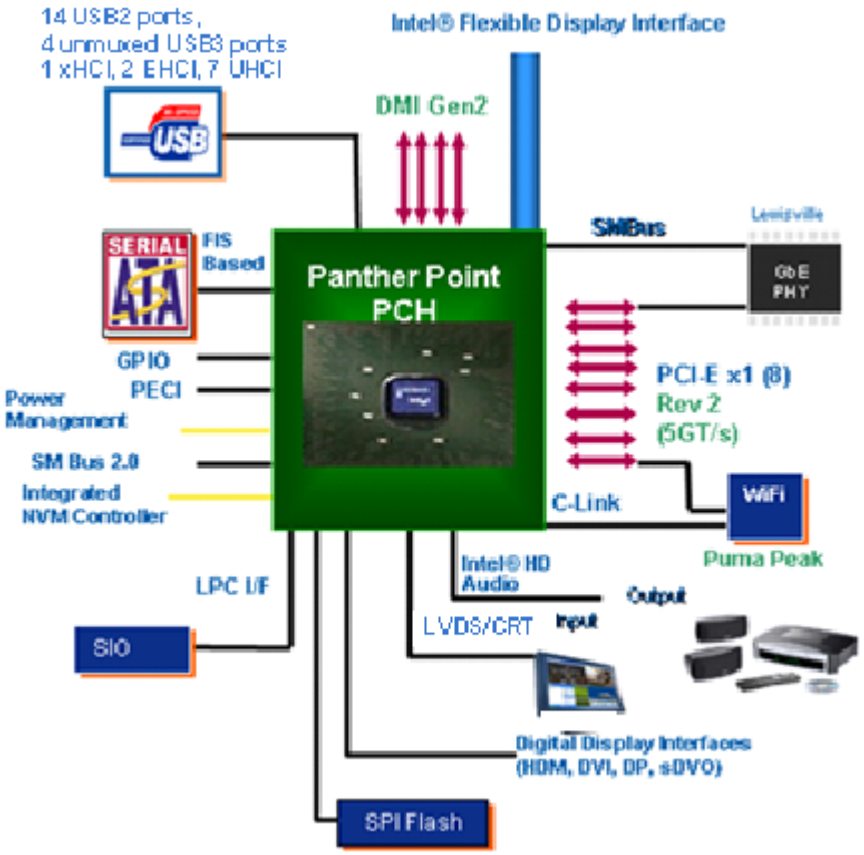


Figure 2.2: PCH Interfaces

2.3 Platform Software Architecture Overview

2.3.1 Firmware

The definition of firmware on PC is instructions (with data) that are consumed by non-IA execution engines associated with a non-CPU hardware device. There are three main categories of firmware:

- Fixed embedded firmware: contained in ROM and hidden from platform view.
- Upgradeable embedded firmware: contained in built-in non-volatile memory with default image (code/data); upgradable during life cycle.

- Externally stored firmware: storage of the code/data is outside of the device package that executes the firmware. The external storage is likely in non-volatile memory form. Patches for embedded firmware falls into this category.

No action is required on platform SW to support fixed embedded firmware. Examples of embedded firmware components are:

- CPU microcode, uncore firmware
- ME ROM code

Upgradeable embedded firmware is presumed to be functional at platform build time. Discrete graphics card firmware is in this category. Upgrade tool is expected to be available for at least one of the user's SW environment. There is no known ingredient with firmware in this category.

2.3.2 Software Stack Overview

The software stack of the modern operating system has also become complex, just like its hardware counterpart. The diagram below depicts how various software execution environments relate to each others. At platform level, today's software execution environment is more than just BIOS and OS applications.

The software stack consists of BIOS over which Operating system is booted. The BIOS maybe be common for all the Operating system. The drivers that OS boots interacts with BIOS, thus making complex software architecture. The figure 2.3 below refers to the modern day software stack and each and every component is important.

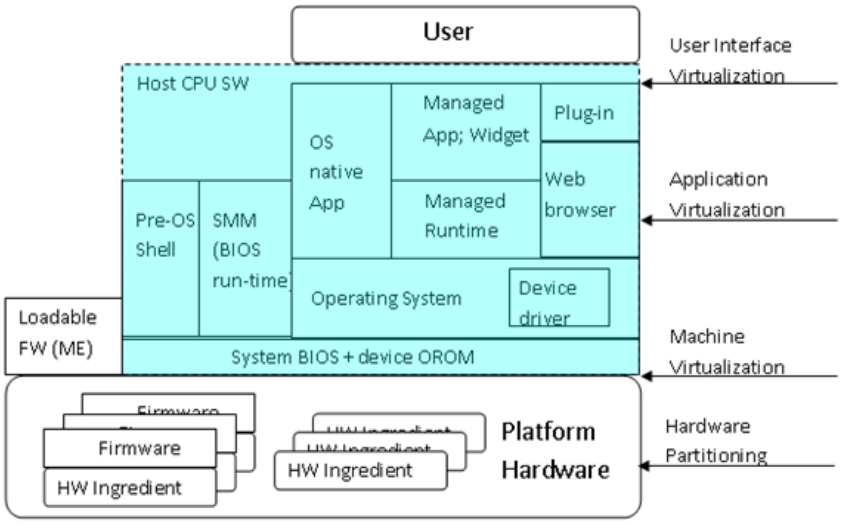


Figure 2.3: Software Stack Overview

2.4 Scope of System BIOS on Platform Validation

BIOS is the first code run by a PC when powered on. As BIOS initialize the various platform components like CPU initialization, Core initialization, memory and chipset initialization etc. it is considered as a main Ingredient in ingredients and platform validation.

2.4.1 System BIOS

The **basic input/output system (BIOS)**, also known as the System BIOS or ROM BIOS, is a de facto standard defining a firmware interface.

The BIOS software is built into the PC, and is the first code run by a PC when

powered on ('boot firmware'). The primary function of the BIOS is to set up the hardware and load and start a boot loader. When the PC starts up, the first job for the BIOS is to initialize and identify system devices such as the video display card, keyboard and mouse, hard disk drive, optical disc drive and other hardware. The BIOS then locates software held on a peripheral device (designated as a 'boot device'), such as a hard disk or a CD/DVD, and loads and executes that software, giving it control of the PC. This process is known as booting, or booting up, which is short for bootstrapping.

BIOS software is stored on a non-volatile ROM chip built into the system on the motherboard. The BIOS software is specifically designed to work with the particular type of system in question, including having knowledge of the workings of various devices that make up the complementary chipset of the system. In modern computer systems, the BIOS chip's contents can be rewritten, allowing BIOS software to be upgraded.

BIOS will also have a user interface (or UI for short). Typically this is a menu system accessed by pressing a certain key on the keyboard when the PC starts. In the BIOS UI, a user can:

- configure hardware
- set the system clock
- enable or disable system components
- select which devices are eligible to be a potential boot device
- Set various password prompts, such as a password for securing access to the BIOS UI functions itself and preventing malicious users from booting the system from unauthorized peripheral devices.

The BIOS provides a small library of basic input/output functions used to operate and control the peripherals such as the keyboard, text display functions and so forth, and these software library functions are callable by external software. In the IBM PC and AT, certain peripheral cards such as hard-drive controllers and video display adapters carried their own BIOS extension Option ROM, which provided additional functionality. Operating systems and executive software, designed to supersede this basic firmware functionality, will provide replacement software interfaces to applications.

BIOS is primarily associated with the 16-bit, 32-bit, and the beginning of the 64-bit architecture eras, while EFI is used for some newer 32-bit and 64-bit architectures. Today BIOS is primarily used for booting a system and for video initialization; but otherwise is not used during the ordinary running of a system, while in early systems (particularly in the 16-bit era), BIOS was used for hardware access - operating systems (notably MS-DOS) would call the BIOS rather than directly accessing the hardware. In the 32-bit era and later, operating systems instead generally directly accessed the hardware using their own device drivers.

The role of the BIOS has changed over time; today BIOS is a legacy system, superseded by the more complex Extensible Firmware Interface (EFI), but BIOS remains in widespread use, and EFI booting has only been supported in Microsoft's operating system products supporting GPT and Linux kernels 2.6.1 and greater builds (and in Mac OS X on Intel-based Macs). However, the distinction between BIOS and EFI is rarely made in terminology by the average computer user, making BIOS a catch-all term for both systems.

2.4.2 Major features of System BIOS

Here is a (not exhaustive) list of the major features in the platform BIOS (not in execution order):

- Core initialization
- CPU initialization (Multi-core, multi-threading)
- CPU microcode update
- Memory initialization (DDR3)
- Chipset initialization (PCI, USB, etc.)
- ME boot handshakes
- BIOS setup and update facilities with protection
- Various pre-OS devices
- User Inputs (keyboard, mouse, PS/2, USB, ...)
- VBIOS (discrete, integrated, switchable)
- MEBx, AMT (Advanced Management Technology)
- RST (was IMSM) OROM, including RAID and Braidwood (NAND)
- Security
- BIOS, HDD password
- UPEK fingerprint sensor
- TXT
- TPM measurement, including iTPM
- DTBx (PBA or DTAM)

- TDT
- Configuration, power and thermal management setup
- ACPI
- APM
- DPPM (Camarillo)
- VT-d tables
- Wake from S-state
- DPST
- Run-time service installation (SMM handler)
- Docking
- Hot keys
- Digital temperature sensors
- Boot select
- Optical drive
- USB
- SATA, SSD
- Boot manager, User OS, EFI Shell, VMM, recovery OS.

2.4.3 Power-on self-test

Power-On Self-Test (POST) refers to routines run immediately after power is applied, by nearly all electronic devices. Perhaps the most widely-known usage pertains to computing devices (personal computers, PDAs, networking devices such as routers, switches, intrusion detection systems and other monitoring devices). Other devices include kitchen appliances, avionics, medical equipment, laboratory test equipment – all embedded devices. The routines are part of a device's pre-boot sequence. Once POST completes successfully, bootstrapping code is invoked.

POST includes routines to set an initial value for internal and output signals and to execute internal tests, as determined by the device manufacturer. These initial conditions are also referred to as the device's state. They may be stored in firmware or included as hardware, either as part of the design itself, or they may be part of semiconductor substrate either by virtue of being part of a device mask, or after being burned into a device such as a Programmable Logic Array (PLA).

Test results may be enunciated either on a panel that is part of the device, or output via bus to an external device. They may also be stored internally, or may exist only until the next power-down. In some cases, such as in aircraft and automobiles, only the fact that a failure occurred may be displayed (either visibly or to an on-board computer) but may also upload detail about the failure(s) when a diagnostic tool is connected. POST protects the bootstrapped code from being interrupted by faulty hardware. Diagnostic information provided by a device, for example when connected to an engine analyzer, depends on the proper function of the device's internal components. In these cases, if the device is not capable of providing accurate information, subsequent code (such as bootstrapping code) may not be permitted to run. This is done to ensure that, if a device is not safe to run, it is not permitted to run.

Chapter 3

BIOS Validation Analysis

As BIOS is identified as an important ingredient of platform , coverage analysis of BIOS validation is presented here to identify major gaps in BIOS validation and recommendations for improving coverage.

3.1 Coverage Matrix

According to product requirement document (PRD)[3] various features are identified and test cases are mapped to relevant features to identify current coverage.Based on current coverage various functional areas are identified where coverage is less and both current coverage level and expected coverage levels are defined.

The following chart 3.1 shows the Coverage matrix to show current BIOS test coverage and expected coverage.

3.2 Coverage Gaps and Test Case Derivation

This section describes the coverage gaps found for BIOS validation and also covers the set of example test cases to fill those coverage gaps.

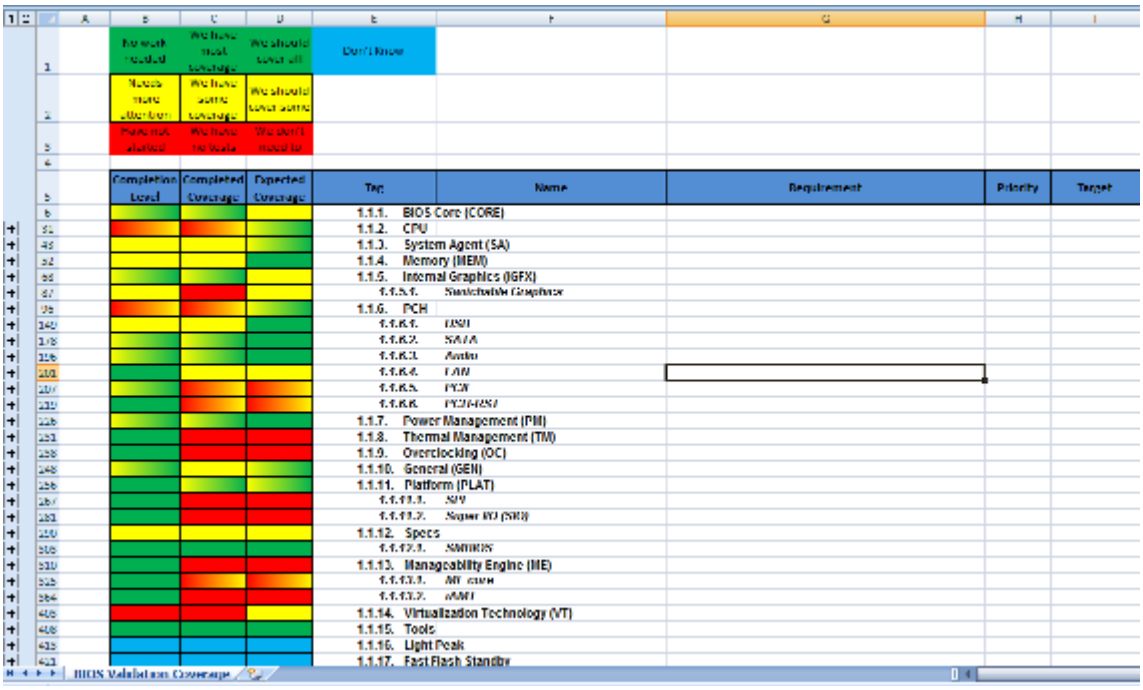


Figure 3.1: Coverage Matrix

3.2.1 MSR Register Level Testing

Machine state register, also known as **model specific register (MSR)** is a feature Intel implemented in their X86 and X86-64 family processors which provides the option to control and receive information regarding the CPU performance. All the MSRs registers handle only system functions and cannot be accessible by application programs.

The CPU uses Read from Model-Specific Register and Write to Model-Specific Register instructions which are used in turn to modify or read from the MSR registers. The operating system usually modifies the MSR registers during the early staging of the boot process.

As a **Example** consider **MSR (1A0h) IA32_MISC_ENABLES** can be used to check **Turbo Mode capability (Turbo Boost Technology)** of the processor.

Bit 38 (Turbo Mode Disable Bit) of MSR 1A0h can be used to check whether Turbo mode is enabled or not. If Bit 38 is 0 then we can ensure that Turbo mode is enabled and if Bit 38 is 1 then we can ensure that turbo mode is disabled.

Example Test Case:

Test case title: Intel Turbo Boost Technology

Test Case Description: Verifies processor Turbo Boost function by enable/disable this feature.

Test Case Procedure:

- Disable Turbo Boost technology for BIOS and verify Bit 38 of MSR (1A0h) is set to 0 or not. Also verify that frequency of the processor is below turbo frequency.
- Now enable Turbo boost technology and verify that Bit 38 of MSR (1A0h) is set to 1 or not. Apply hig load on system and verify frequency of the processor is set to turbo frequency.

Advantage of MSR Level Testing:

- The main advantage of MSR level testing is that it can provide next level of details at the time of debugging.
- Another advantage is MSR level testing is that it can be fully Automated easily.

3.2.2 UEFI (Unified Extensible Firmware Interface):

The **Unified Extensible Firmware Interface (UEFI)** is a specification that defines a software interface between an operating system and platform firmware. UEFI

is a more secure replacement for the older BIOS firmware interface, present in all IBM PC-compatible personal computers, which is vulnerable to bootkit malware.

The original EFI (Extensible Firmware Interface) specification was developed by Intel. In 2005, development of the EFI specification ceased in favour of UEFI, which had evolved from EFI 1.10. The UEFI specification is being developed by the industry-wide organization Unified EFI Forum. UEFI is not restricted to any specific processor architecture and can run on top of, or instead of, older BIOS implementations.

The interface defined by the EFI specification includes data tables that contain platform information, and boot and runtime services that are available to the OS loader and OS. UEFI firmware provides several technical advantages:

- Ability to boot from large disks (over 2 TiB)
- Faster boot-up
- CPU-independent architecture
- CPU-independent drivers
- Flexible pre-OS environment, including network capability
- Modular design

Some existing enhancements to PC BIOS, such as the Advanced Configuration and Power Interface (ACPI) and System Management BIOS (SMBIOS), are also present in EFI, as they do not rely on a 16-bit runtime interface.

Services

EFI defines two types of services: boot services and runtime services. Boot services are only available while the firmware owns the platform (before the "ExitBootServices" call). Boot services include text and graphical consoles on various devices and bus, block and file services. Runtime services are still accessible while the operating system is running; they include services such as date, time and NVRAM access.

Booting

UEFI does not rely on a working boot sector only, but needs a special partition table referring to a special partition containing a specially located file with a standardized name depending on the actual architecture to boot ([architecture name].efi). Boot loaders are a class of UEFI applications. As such, they are stored as files on a file system that can be accessed by the firmware. Boot variables, stored in NVRAM, indicate the paths to the loaders. Boot loaders can also be auto-detected by firmware, for instance to enable booting on removable devices.

It is common for UEFI firmware to include a boot manager, to allow the user to select and load the operating system among the possible options.

The EFI shell

EFI provides a shell environment.The shell can be used to execute other EFI applications.
As UEFI is becoming more and more popular and also UEFI is must for latest operating system Microsoft Windows 8 to support "UEFI Secure Booting", it is necessary

to validate complete BIOS settings and Platform under UEFI shell.

In EFI shell using different commands we can check whether correct BIOS settings are populated or not.

Example Test Cases

- **Test Case 1:** Using "memmap" command we can verify whether system reports correct memory count under EFI shell or not.
- **Test Case 2:** Using "PCI" command we can verify whether system reports all connected PCI-E devices under EFI shell or not.
- **Test Case 3:** "dmpstore" command can be used to verify all NVRAM variables.
- **Test Case 4:** "Drivers" command can be used to verify list of drivers that follow EFI driver model.
- **Test Case 5:** "smbiosview" command can be used to verify whether correct hardware information is populated in SMBIOS tables or not.

3.2.3 Port 80h POST Codes

During the **Power-On Self Test (POST)**, the BIOS sends progress codes (POST codes) to I/O port 80h. If the POST fails, the last POST code generated is left at port 80h. This code can be used to find out why the error occurred.

Typical Port 80h POST Sequence:

Port 80h code values typically increase during the boot process. The early codes are for subsystems closer to the processor and the later codes are for peripherals. Generally, the order of initialization is Processor,Memory,Busses,Output/Input Devices,Boot Devices. The sequence of POST is system-specific.

All post codes are very well defined but we should manually generate failure and verify that whether BIOS is sending correct POST error codes or not at time of failure.

Example Test Cases

Test Case Title: No Memory

Test Case Description: Verifies that the system does not boot and gives the correct beep codes and POST Codes when no memory is present in the system

Test Case Procedure:

- Power on the system with no memory located in any of DIMM slot.
- Verify that system emits three consecutive beeps and fails to boot.
- Check POST code LEDs shows DD53(for Mobile Board) and 53 (for Desktop board).
- Now install proper memory in each slots and verify system boots properly.

Advantage of POST Code Level Testing

- It can also provide next level of information at the time of debugging.

3.2.4 SMBIOS

System Management BIOS (SMBIOS) specification defines data structures (and access methods) in a BIOS which allows a user or application to store and retrieve information specifically about the computer in question.

The SMBIOS Specification addresses how motherboard and system vendors present management information about their products in a standard format by extending the BIOS interface on x86 architecture systems. The information is intended to allow generic instrumentation to deliver this information to management applications that use DMI, CIM or direct access, eliminating the need for error prone operations like probing system hardware for presence detection.

This specification is intended to provide enough information that BIOS developers may implement the necessary extensions to allow the hardware on their products and other system-related information to be accurately determined by users of the defined interfaces. In addition, in cases where the implementer has provided write access to non-volatile storage on the system, some information may be updated by management applications after a system is deployed in the field to record data that persists between system starts.

SMBIOS contains different types of Table as listed below and according to type of table hardware information is populated in that table.

- Type 0 BIOS Information
- Type 1 System Information
- Type 2 Mainboard Information
- Type 3 Enclosure/Chasis Information

- Type 4 Processor Information
- Type 7 Cache Information
- Type 9 System Slots Information
- Type 16 Physical Memory Array
- Type 17 Memory Device Information
- Type 19 Memory Mapped Device Mapped Address's
- Type 32 System Boot Information

We must verify whether correct hardware information is populated into SMBIOS tables or not.

Example Test Cases:

- **Test case 1:** Verify that the SMBIOS Type 0 structure BIOS ID string coincides with that of the BIOS string that is on the system.
- **Test Case 2:** Verify that the SMBIOS Type 2 structure Product Name String coincides with BIOS Product Specification.
- **Test Case 3:** Verify whether correct memory details are populated in SMBIOS structure type 16, type 17 and type 19 or not.

3.2.5 Advance Configuration and Power Management (ACPI) Tables Verification

The Advanced Configuration and Power Management Interface (ACPI) specification [4] contains interfaces that provide standard controls and operation needed to perform system and device power management. This information is most useful for operating

system vendors, OEMs and IHVs.

In computing, the Advanced Configuration and Power Interface (ACPI) specification provides an open standard for device configuration and power management by the operating system. ACPI aims to consolidate and improve upon existing power and configuration standards for hardware devices. It provides a transition from existing standards to entirely ACPI-compliant hardware, with some ACPI operating systems already removing support for legacy hardware. With the intention of replacing Advanced Power Management, the Multiprocessor Specification and the Plug and Play BIOS Specification, the standard brings power management under the control of the operating system (OSPM), as opposed to the previous BIOS central system, which relied on platform-specific firmware to determine power management and configuration policy.

The ACPI specification contains numerous related components for hardware and software programming, as well as a unified standard for device/power interaction and bus configuration. As a document that unifies many previous standards, it covers many areas, for system and device builders as well as system programmers.

As ACPI maintains different tables we must verify all tables are compliance with Specification or not.We must develope set of test cases to verify different ACPI tables like **DSDT,ECDT,FACS,FADT,HPET,MADT,MCFG,RSD,PTR,RSDT** must be compliance with ACPI Specification and wether correct information is populated in those tables or not.

Chapter 4

Platform Stress Analysis

4.1 Stress Testing

Stress testing is a form of testing that is used to determine the stability of a given system or entity. It involves testing beyond normal operational capacity, often to a breaking point, in order to observe the results.

4.1.1 Stress testing in terms of Hardware

Reliability engineers often test items under expected stress or even under accelerated stress. The goal is to determine the operating life of the item or to determine modes of failure.

Stress testing in general, should put the hardware under exaggerated levels of stress in order to ensure stability when used in a normal environment.

Example

Computer Processors

When modifying the operating parameters of a CPU, such as in overclocking, un-

derclocking, overvolting, and undervolting, it may be necessary to verify if the new parameters (usually CPU core voltage and frequency) are suitable for heavy CPU loads. This is done by running a CPU-intensive program (usually Prime95) for extended periods of time, to test whether the computer hangs or crashes. CPU stress testing is also referred to as torture testing. Software that is suitable for torture testing should typically run instructions that utilize the entire chip rather than only a few of its units.

Stress testing a CPU over the course of 24 hours at 100% load is, in most cases, sufficient enough to determine that the CPU will function correctly in normal usage scenarios, where CPU usage fluctuates at low levels (50% and under), such as on a desktop computer.

4.2 Load Testing

Load testing is the process of putting demand on a system or device and measuring its response. Load testing is performed to determine a system's behavior under both normal and anticipated peak load conditions. It helps to identify the maximum operating capacity of an application as well as any bottlenecks and determine which element is causing degradation. When the load placed on the system is raised beyond normal usage patterns, in order to test the system's response at unusually high or peak loads, it is known as stress testing. The load is usually so great that error conditions are the expected result, although no clear boundary exists when an activity ceases to be a load test and becomes a stress test.

There is little agreement on what the specific goals of load testing are. The term is often used synonymously with software performance testing, reliability testing, and volume testing. Load testing is a type of non-functional testing.

4.2.1 User Experience under Load test

User Experience under Load test In the example above, while the device under test (DUT) is under production load - 100 VUsers, run the target application. The performance of the target application here would be the User Experience under Load. It describe how fast or slow the DUT responds, and how satisfied or how the user actually perceives performance.

So as User experience is becoming more and more important we will try to focus on user experience under specific load/stress conditions.we will check that whether system is behaving properly or not in specific load/sterss conditions.

4.3 Existing Stress Test Content Analysis

Example Set of Stress Test cases to be run overnight:

- S3(Sleep) state - 1000 cycles
- S4(Hibernet) state - 1000 cycles
- S5(system-off) state - 1000 cycles
- FFS(Fast Flash Standby) state - 1000 cycles
- Deep S3 state - 1000 cycles
- Deep S4 state - 1000 cycles
- Deep S5 state - 1000 cycles
- Processor Stress Test
- TAT (Thermal Analysis tool) 100

- 3D Mark Graphics Benchmark tests
- PC Mark System performance Benchmark Test
- Memtest (memory stress test)
- HDDs (SATA) Stress
- USB stress test

4.4 Problem Statement Based on Existing Stress Content Analysis

- All existing stress test scenarios are Ingredient centric than Platform centric. It means that stress test cases are not written and executed considering whole platform or they are more focused on specific feature/capability or Ingredient.

  For example if you consider Processor stress test, memory stress test, USB stress test or SATA stress test they all are targeting specific Ingredient or Specific Interface.

- As we can see from above test cases, we are doing rigorous Power management testing (S3, S4 and S5 cycling 1000 times) for complete platform stress. But it's an incorrect assumption as we have to consider all platform components and their impact on power management. We have to check impact of each and every Ingredient on power management and also impact of power management on performance of each and every Ingredient.

- As far as success criteria (Pass/Fail criteria) for stress test is concerned, our success criteria is just like there should be no error, No hang and No BSOD (Blue

Screen Of Death) during stress testing.But there must be a systemic measure of stress to identify impact of stress on the system. We must identify more parameters (like performance parameters) that can be measured and documented as a part of success criteria.

- Lack of Standardized and more advanced tools for stressing Platform components.

Chapter 5

Background Into Methodology

This chapter will describe the methodology developed for stress evaluation for different platform Componets/Interfaces.

5.1 Performance Counters

From [5], At the heart of the **CHAP counters [Chipset Hardware Architecture performance counters]** functionality are counters, each with associated registers.

Each counter has a corresponding command, event, status, and data register. The smallest recommended implementation will have 2 counters, but if justified for a particular product, this architecture can support many more counters. Typically 8 CHAP counters are in an PCH. The primary consideration is available silicon area. The memory mapped space currently defined can accommodate registers for 256 counters. It could be configured for more, but that is beyond what is currently practical.

Signals representing events from throughout the chip are routed to the CHAP unit. Software can select events that will be recorded during a measurement session. The number of counters in an implementation defines the numbers of events that can

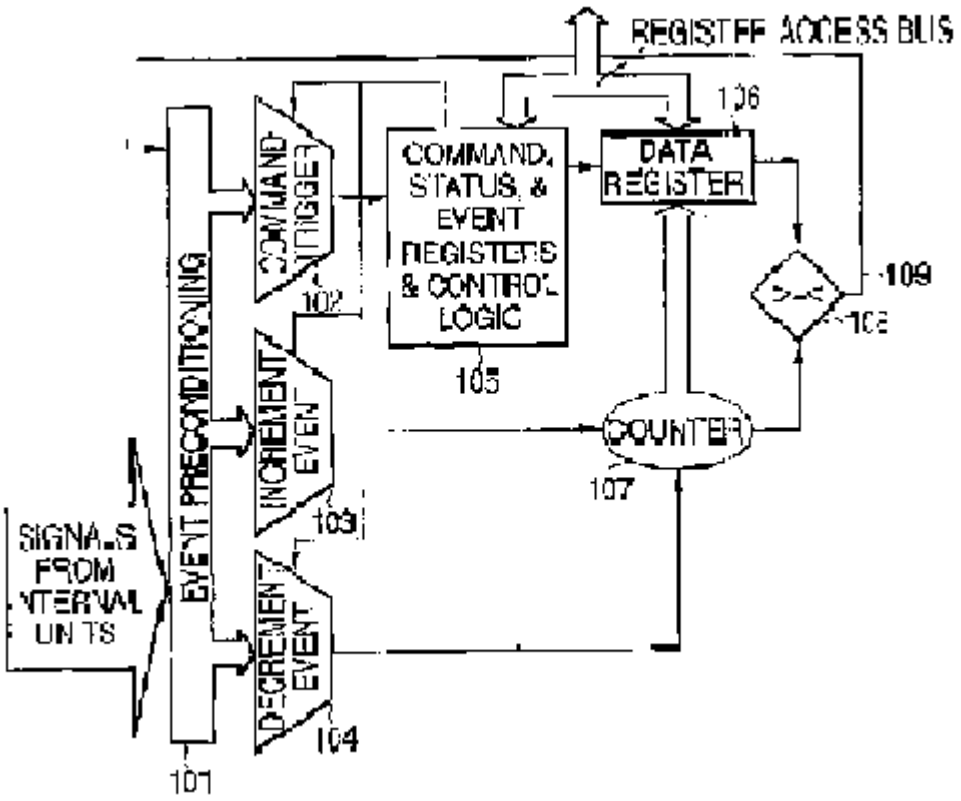


Figure 5.1: Performance Counter[6]

be recorded simultaneously. Software and hardware events can control the starting, stopping, and sampling of the counters. This can be done in a time-based (polling) or event-based fashion. Each counter can be incremented or decremented by different events. In addition to simple counting of events the unit can provide data for histograms, queue analysis, and conditional event counting (**Example:** How many times did event A happen before the first event B took place).

When a counter is sampled, the current value of the counter is latched into the corresponding data register. The command, event, status, and data registers are accessible via standard PCI memory mapped registers in order to facilitate high-speed sampling. This unit is a Plug-and-Play PCI compliant device with a base address scheme for its memory mapped space.

5.2 Stress Evaluation using Performance Counters

CHAP counters can be programmed to track/count number of events. So first step is to identify the set of events that an be tracked. Limited events will be available and also events must be selected according to Interface/component under test. Second step will be to configure/program CHAP counters to track those events.As command, event, status, and data registers are accessible via standard PCI memory mapped registers , we need to program/configure event registers to track specific events.

Next step will be to start the target counters to track those events, and for that we need to configure command registers.

Once CHAP counters are configures to track events we should generate those events so that we can track them.In our case after configure and starting counters we should apply stress on Interface so that we can measure how much stress is applied on Interface.

Now last step will be to sample the counters and read the event counts during specific Interval[**Time based Sampling**][**6**] or read the event counts when some other event occurs [**Event based sampling**][**6**].

Summary of Steps :

- **Step 1 :** Identify which events are important for test (number of reads , number of writes , number of interrupts , Cache hits/miss etc.)
- **Step 2 :** Program/Configure CHAP counters to track those events (configure event registers)
- **Step 3 :** Start the counters to track those events (configure command register)
- **Step 4 :** Generate those events which we want to track

- **Step 5 :** Sample the counters and read the event counts from data registers(configure data registers)

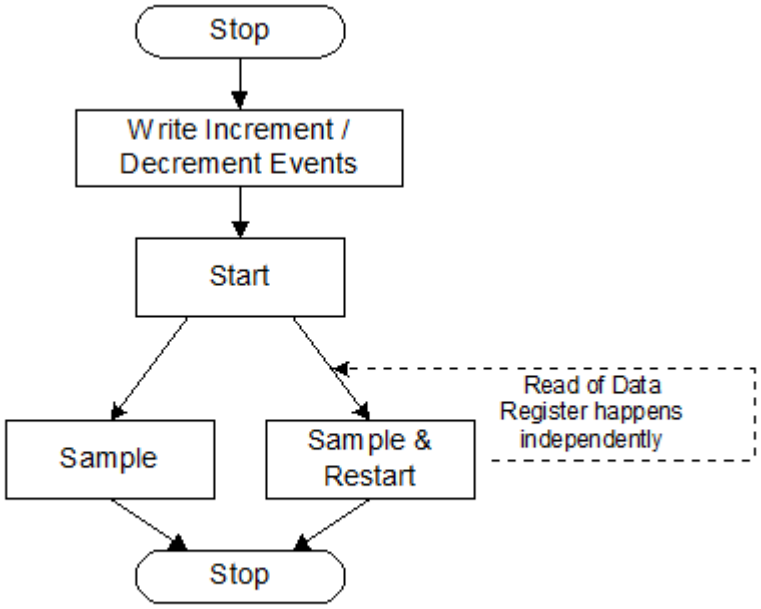
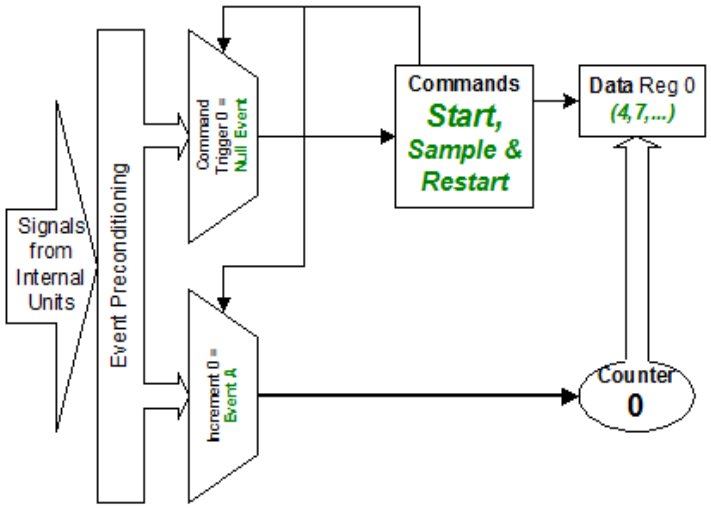


Figure 5.2: Command Flow[5]

5.3 Time Based Sampling Example[5]

The following example 5.3 has been simplified by using 12 clocks as the sampling period. In a real system the sampling would more likely be something like 1 ms. There is a certain amount of overhead associated with writing and reading to any CHAP registers. The more frequent the interaction between the CHAP counters and any software, the larger the margin of error that will be injected into the final results.



Opcode	Target Counter	Increment Event	Decrement Event	Trigger Event
Write Event Register	0	Event A	None (000h)	
Start	0			Immed (000h)
* Sample & Restart	0			Immed (000h)
Read Data Register	0			
-- Wait 12 clocks before returning to * --				

Figure 5.3: Performance Counter Structure

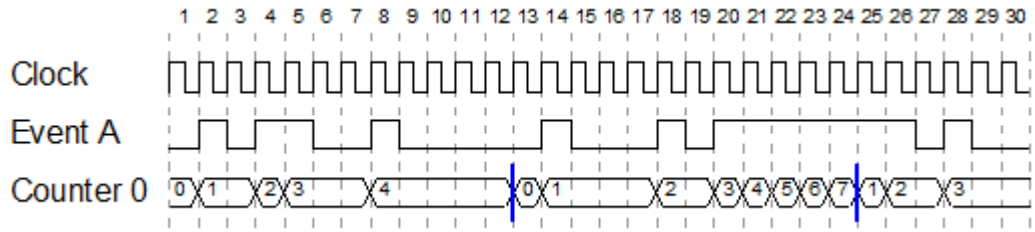


Figure 5.4: Time Based Sampling Example

The sampling period can be controlled by a CPU counter or another CHAP counter periodically interrupting the system and allowing the software to read data registers or do whatever else may be desired.

The table above 5.3 demonstrates: How many times did event A occur during 12 clocks?

An alternative way to represent the data in the preceding table is as follow:

Write Event Register 0 (Increment = Event A)
Start Counter 0 immediately
Repeat every 12 clocks
Sample Restart Counter 0 (Threshold Condition Code is NA)
Read Data Register 0
End Repeat

5.4 Stress Evaluation of SATA Interface using Methodology

Following are the steps for SATA interface stress evaluation using our methodology.

Step 1: For Stress analysis on SATA Interface our interest of events will be number of read data transfer , number of write data transfer , total data transfer on SATA Interface.

Step 2 :

- Configure counter 0 for total no of data transfer
- Configure counter 1 for total no of Read data transfer
- Configure Counter 2 for total no of Write data transfer

Step 3 : Start all the counters

Step 4 : Apply Workload/Stress on SATA Interface [Large file copy transfer / Use stress tools]

Step 5 : Sample and Read the data registers of each counters every second to count actual data transfer rate(MBps) on SATA Interface

Chapter 6

Results

6.1 Case Study : SATA Interface

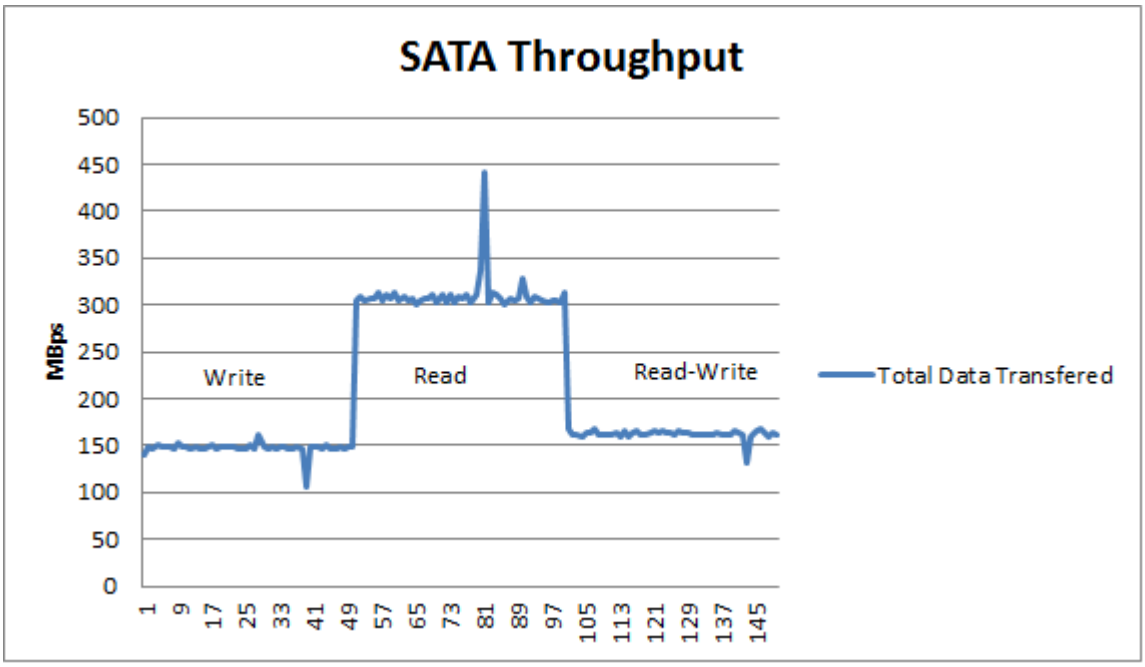


Figure 6.1: SATA Throughput when SATA is stressed with IOmeter

Above study on SATA Interface shows that different types of traffic generates different level of Stress on interface.Different type of traffic is generated on SATA

interface using IOmeter I/O stress tool. Different types of traffic includes,

- Read traffic
- Write traffic
- Read-Write traffic.

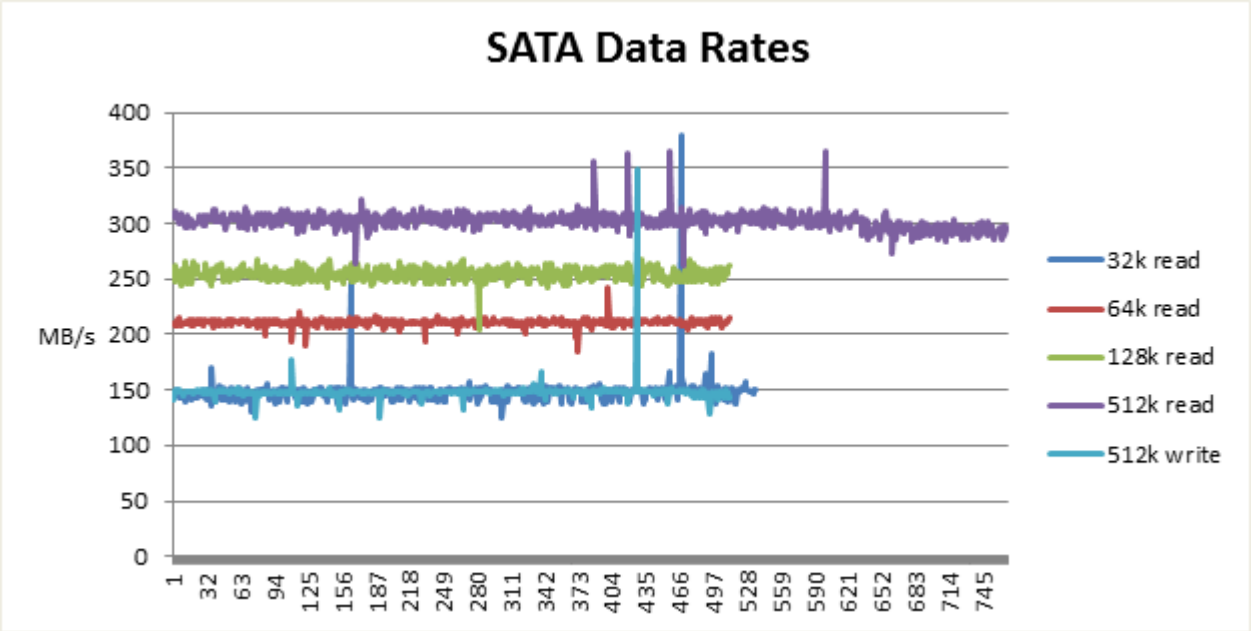


Figure 6.2: SATA Throughput when SATA is stressed with IOmeter with different Access Specifications

Also as per above study different Access Specifications generates different level of stress on interface,

- Transfer Request size (4k, 8k ,12k ,32k,..)
- Random traffic
- Sequential traffic

So we must check level of stress applied on interface with different Access Specifications (4k ,8k ,16k , Random , Sequential etc.) and choose access specification accordingly to stress interface effectively.

6.2 Case Study : SATA and USB Interface

The following study shows the impact of stress on SATA and USB Interface when all other platform components (Memory, Graphics,CPU, LAN etc.) are heavily stressed. Based on this type of analysis we can build effective stress scenarios which can effectively stress all platform components and find defects not detected under normal stress condition.

The below Study shows the SATA and USB throughput when all other platform components are heavily stressed.

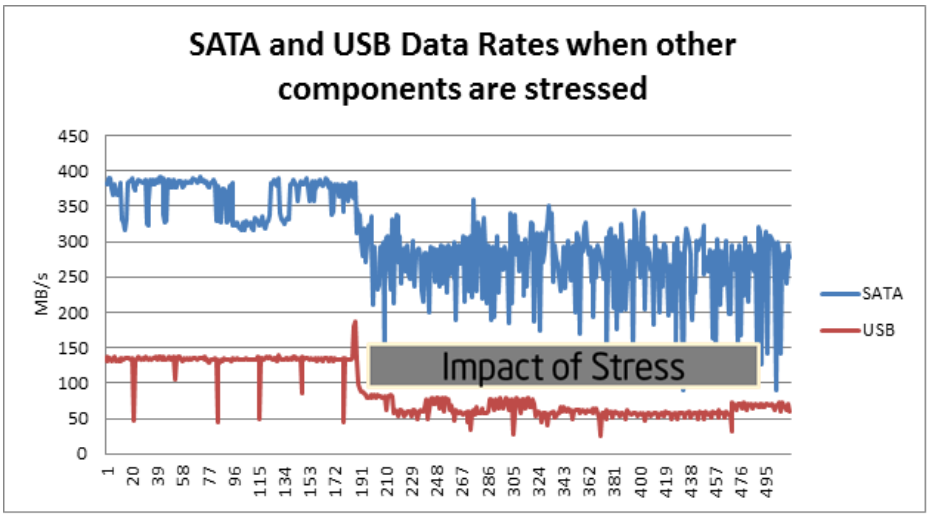


Figure 6.3: SATA and USB Throughput when other Components are Stressed

6.3 Case Study : PCIe Interface

The following cases study shows how this methodology can be used for tools characterization. Different tools are available for stressing different platform Components/Interfaces and as we know different tools stress componets differently. So there must be a way to figure out amongst all which tool effectively stress component/interface.

The below study shows that amongst all LAN stressing tools NTTTCP tool effectively stress LAN Interface. Based on this type of study we can easily figure out which tool can be used for stressing component/interface effectively.

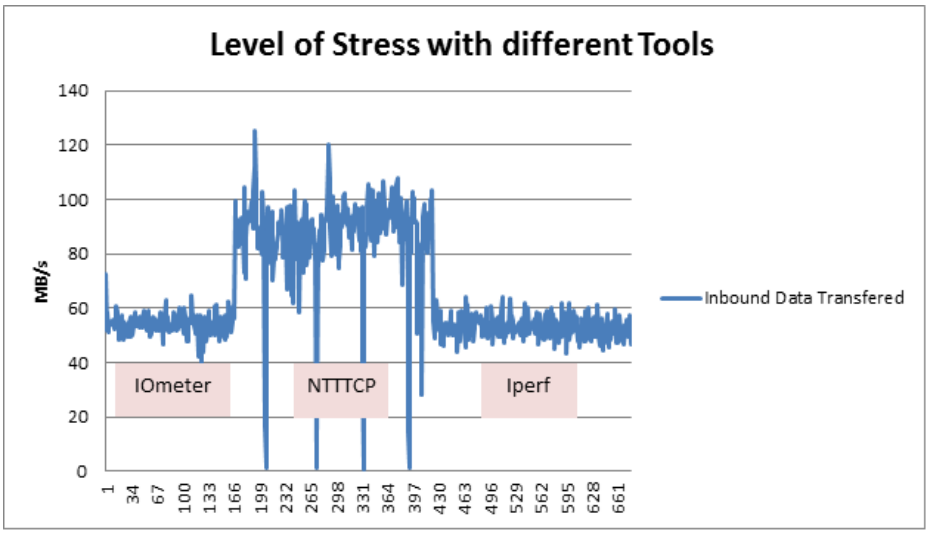


Figure 6.4: LAN (via PCIe)Throughput

Chapter 7

Recommendation and Approach

- Instead of stressing each Ingredient or component individually (Ingredient centric testing) develop stress scenarios to exercise concurrency of multiple ingredients/components.
- Based on this type of analysis build stress scenarios which will effectively stress each platform components/interfaces.
- While Stressing each components/interface use Tools/Access Specifications which generates Maximum Stress on Interface.
- Also use automated framework to port all the tools and define logic in executing the stress test.(Parallel,Sequential,Random etc.)

Chapter 8

Conclusion and Future Scope

8.1 Conclusion

Our study shows that different Tools or Access Specifications stress platform components differently and at different level.By using this type of methodology we must recommend and also develop Tools which can effectively stress platform components and find marginal defects not detected under normal stress condition.

Our cases study shows that we can also identify architectural inefficiency of the Component/Interface at protocol or hardware level.

8.2 Future Scope

Similar studies needs to be carried out for other platform Components/Interfaces like Memory , Graphics , CPU etc. to measure the actual impact of stress on component/interface and based on that effective stress scenarios can be created.

The overall objective should be to test all platform components effectively and find architectural defects that are not detected under normal testing.

References

[1] Intel Developers,Intel Platform and Component Validation - Whitepaper,Intel Technology India Pvt. Ltd.

[2] Intel Developers,2012 client Platform Architecture Document(PAD),Intel Technology India Pvt. Ltd.

[3] Intel Developers,2012 client Product Requirement Document(PRD),Intel Technology India Pvt. Ltd.

[4] http://www.acpi.info/spec.htm

[5] Intel Developers , CHAP Counters RAS 1.5 , Intel Technology India Pvt. Ltd.

[6] James S. Chapple , Hardware Event Based Flow Control of Counters , Intel Technology India Pvt. Ltd. , US Patent US6519310 B3 , FEB 11 ,2003

[7] Intel Developers, RS-IvY Bridge Processor Family BIOS Writers Guide(BWG),Intel Technology India Pvt. Ltd.

[8] http://en.wikipedia.org/wiki/Platform_Controller_Hub

[9] http://en.wikipedia.org/wiki/BIOS

[10] http://en.wikipedia.org/wiki/Model-specific_register

[11] http://en.wikipedia.org/wiki/SMBIOS