Developing a framework for Power Management Validation on Windows 8 OS

By

Bhavesh Gopani

Roll No: 10MECV06

Department of Electronics and Communication Engineering

Institute of Technology

Nirma University

Ahmedabad-382481

May 2012

Developing a framework for Power Management Validation on Windows 8 OS

Major Project Report

Submitted in partial fulfillment of the requirements

For the degree of

Master of Technology

In

Electronics And Communication Engineering

(VLSI Design)

By

Bhavesh Gopani

(10MECV06)

Under the Guidance of

Ms. Suganthi B.

Intel Technology India Pvt. Ltd.

Department of Electronics and Communication Engineering

Institute of Technology

Nirma University

Ahmedabad-382481

May 2012

Declaration

This is to certify that

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

ii) Due acknowledgement has been made in the text to all other material used.

Bhavesh Gopani

Certificate

This is to certify that the Major Project entitled **" Developing a framework for Power Management Validation on Windows 8 OS"** submitted by **Bhavesh Gopani (10MECV06)**, towards the partial fulfillment of the requirements for the degree of **Master of Technology in VLSI Design** of **Nirma University of Science and Technology, 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.

Date: ____________________	**Place: Ahmedabad**
External Guide	**HOD**
____________________	____________________
Ms. Suganthi B.	**Prof. A. S. Ranade**
Intel Technology India Pvt. Ltd.,	**Professor, EC**
Director	**Internal Guide**
____________________	____________________
Dr.K.R.Kotecha	**Prof. Usha Mehta**
Director, IT, NU,	**VLSI Design,EC Department**
	Nirma University

Acknowledgements

Apart from the efforts of me, the success of this project depends largely on the encouragement and guidelines of many others. I take this opportunity to express my gratitude to those people who have been instrumental in the successful completion of this project.

I would like to thank my manager **Mr. S. Siddanagouda**, **Engineering Manager**, Platform Validation and Debug Team, Intel Technology India Pvt. Ltd., Bangalore, for giving me this project and for his guidance at each and every stage of the project even in his busy schedule.

I thank my **mentor Ms. Suganthi B.**, Team Lead,Platform Validation Team, Intel Technology India Pvt. Ltd, who helped me a lot in framing out my project, for her timely guidance, support and also allowing me to have faithful discussions throughout the project work.

I would like to thank Platform Validation Team members at Intel Technology India Pvt. Ltd., **Mr.Shanmuganathan**, **Mr.K.M.Sivaprakash** **Mr.Anand**, **Mr.Pavan**, **Mr.Santosh** and **Mr.Gurukumar** for their excellent direction and guidance for my project work. In addition, a special thanks to Management of Intel Technology India Pvt. Ltd, Bangalore for providing me with all the required facilities.

I would like to express my sincere thanks to **Dr. Ketan Kotecha, Director**,Nirma Institute of Technology and **Dr.N M Devashrayee,PG Coordinater**,VLSI Design,Nirma University for their kind patronage.

I would like to thank my internal guide **Prof.Usha Mehta** for guiding, constant encouragement, support and suggestions for improvement during the course of my project. His encouraging words brought out the best in me.

- Bhavesh Gopani
10MECV06

Abstract

Today, the complexity of computing platforms has increased with the development of SoC' s coupled with dozens of platform component and support for multiple Operating systems. Validation of SoC and platform component is immensely complicated. One of the most important features of a Platform is Power Management and Performance of system. Power management aims at reducing operating costs for energy and cooling systems while at the same time keeping the performance of a system at a level that matches the current requirements. Thus, power management is always a matter of balancing the actual performance needs and power saving options for a system. Power management can be implemented and used at different levels of the system. A set of specifications for power management functions of devices and the operating system interface to them has been defined in the Advanced Configuration and Power Interface. As power savings in tablet platform can primarily be achieved on processor level, system level, platform level .The aim of this project to develop a tool which is use to measure attributes contributing to Power and Performance on SoC and other peripheral devices on 32nm based SoC Tablet Platform with supporting Windows 8 Operating System. The features supported by this tool are C-state Residency, P-State Residency, and S-State Residency etc.

Contents

List of Figures

Chapter 1

Introduction

1.1 Problem Definition

Nowadays, Computing Technology is growing rapidly. The Consumer demands miniature version of computer or hand held devices which are capable of running mobile application and performing computing operation, supporting multiple Operating Systems with virtual touch interface called Tablet PC. Unlike laptops, these handheld devices have limited resources like CPU, memory and battery life. As features increases, Platform architecture and SOC architecture become more and more complex. User needs multiple features and longer battery backup. So the main challenge is to optimize power consumption. These can be done by innovations in Power management techniques. Intel Architecture supports different power management techniques at different level. For example CPU level, SoC level, Platform level. So validation also becomes important at platform level to determining the proper functionality of design. My objective is to develop a tool for measuring attributes to power management states on 32nm SoC based platform with supporting windows 8 operating system.

1.2 Purpose

The purpose of this tool to validate power management states supported by platform and need for validate the power management techniques' compatibility with windows 8 operating system. There is a high demand to complete the validation procedure in a short period of time with high quality to find almost all the bugs on the platform before the product release. Time-to-market should be minimized so that the company can compete in the market as early as possible. The customer needs are also changing at a faster pace, so the challenge for every company is to launch the latest technology to the market at the earliest.

1.3 Motivation

Intel Corporation has been in an unique position of designing and manufacturing microprocessors that are optimized along the three axes of performance, manufacturing cost, and architecture compatibility. Company is entering the mobile platform market now with their first Tablet Platform with supporting windows 8 operating system code named Clovertrail. There is a high demand for the company to launch the Tablet platform to the market as early as possible to compete with other companies which uses ARM architecture for their Tablets. The proposed framework helps the company to speed up the validation procedure and deliver the product of superior quality which outperforms the similar product from its competitors.

1.4 Literature Survey

INTEL has been in a unique position of designing and manufacturing microprocessors that are optimized along the three axes of performance, manufacturing cost, and architecture compatibility. Achieving optimum balance along any of the two axes is a simpler task than across all three. Successfully meeting these requirements across many generations has resulted in complex designs that impose severe validation requirements on the processors. Some of the important issues unique to Intel that necessitate rigorous validation are the high volumes of IA-32 microprocessors that are shipped and used in a very diverse set of applications. Such diversity implies that the chance of a processor hitting an untested corner case is relatively high compared to a microprocessor used in a restricted set of server applications. The high volume of microprocessors shipped implies that the cost of a bug escape is extremely high in loss of customer revenue and reputation of product reliability. Thus, Intel puts an extraordinary amount of effort in ensuring the processor is thoroughly validated.

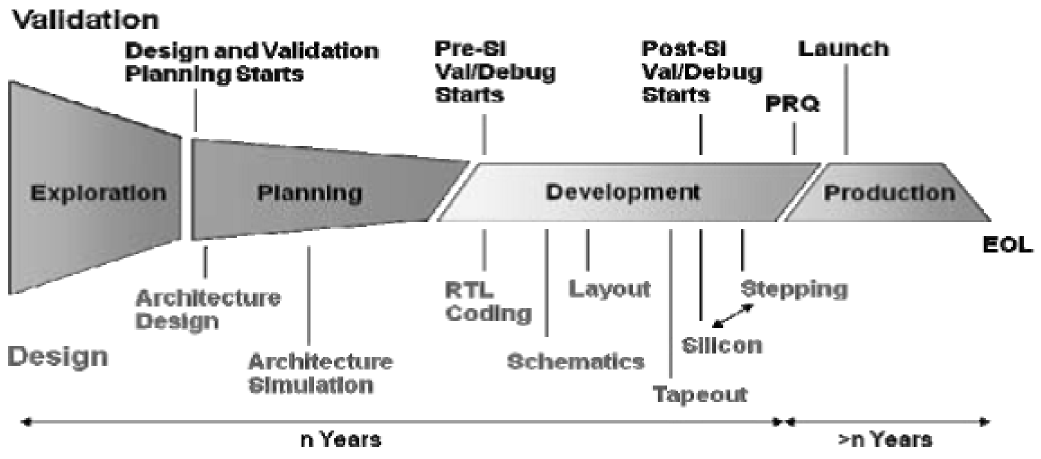


Figure 1.1: Design and Validation Life cycle

Intel has tackled the validation challenge in ways that include expanding the scope of validation in presilicon and postsilicon, making older techniques more efficient, validating across as many levels of abstraction as applicable to break the problem's complexity, and innovating better techniques to address complexity of new micro architectural features. Validation techniques may be broadly classified as presilicon and postsilicon. Most of the presilicon validation is done either on a register transfer-level simulator or an emulator and has been the primary technique successfully used by a number of commercial manufacturers. After the first silicon is ready, postsilicon validation is done in a system environment to flush out bugs missed in presilicon validation. Typical reasons for this are the slow simulation speed that prevents running a large number of tests, tests requiring long execution times, testing done close to release in a hasty manner, tests not run in a particular mode, innovations in circuit technology, and so forth.

Typical postsilicon errors are from interactions between different components in very improbable corner cases; MIPS R4000, data showed that such interactions consisted of 56.5

Post-silicon validation and debug 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. Chips comprising 500,000 logic elements are the silicon brains inside cell phones, MP3 players, computer printers and peripherals, digital television sets, medical imaging systems, components used in transportation safety and comfort, and even building management systems. Either because of their broad consumer proliferation, or because of their mission-critical application, the manufacturer must be absolutely certain that the device is thoroughly validated. Postsilicon Validation Methods Typical postsilicon methodology consists of booting low-level console user interface, running legacy tests, doing a boot of the favorite operating system, running presilicon tests, running postsilicon tests, locating and diagnosing bugs, reproducing and analyzing bugs on the RTL model, and using microcode patches to provide a work-around so that validation may proceed. This is typically done with low observability using the small amount of DFT features available on the chip. Intel's postsilicon validation effort is done along two major directions: system validation and compatibility validation. System validation is primarily focused on validating the CPU and chip sets in an embedded systems environment that uses the new silicon in a multiway configuration and has special monitor software to download the validation tests into the platform for execution on bare silicon. The embedded platform contains glue logic to provide a flexible system configuration with chip sets that is representative of a real system. This platform has many hooks to provide flexibility in testing and has external graphics and Peripheral Component Interconnect peripherals that can communicate with the CPU core and generate programmable traffic on the front-side bus.

The system validation tests that run on this platform are targeted toward finding CPU and chip set interface problems. Compatibility validation runs on desktop and server systems that incorporate the new chips with various configurations of the system and are run under real operating systems, other system software, and user applications. Validation of circuit problems is distributed across the groups but may also be handled by a separate group.

System validation attempts to find bugs in the processor not only for common usage models but for uncommon usage as well if the architecture permits. Compatibility validation, on the other hand, uses real applications and thus validates common usage models. System validation tests are self-contained and are easier to debug when they find a problem as compared to problems found with compatibility tests because of the complexity of the system and application software.

Chapter 2

Intel Platform and Component Validation

2.1 Introduction

The internal complexity of the Tablet platform has grown to a staggering level. Today's most advanced SoC and platform component and must be compatible with dozens of operating systems, software applications. To ensure leading performance, reliability and compatibility in this complex environment, Intel invests over millions dollars annually in component and platform validation.

Intel's validation process begins during the first stages of component design - and continues throughout pre-silicon and post-silicon development. All core platform components are exhaustively tested, both independently, and with an enormous variety of third-party hardware and software components. Operating conditions and performance demands are pushed to extreme limits. Issues are unearthed and resolved, and the findings are used to drive constant improvement in Intel's design and manufacturing processes. Intel's design and validation engineers also work closely with leading hardware and software vendors throughout the industry to coordinate development and testing efforts. This helps to ensure that new technologies are integrated quickly and reliably into next-generation products, so users can rely on the highest level of platform functionality. The results of this unprecedented commitment to quality are better products and faster time to market, important advantages for the many businesses and users who depend on the performance, reliability and compatibility of Intel platforms and components.

2.2 The Importance of Validation

Tablet PC reliability and compatibility issues can be costly for users. For businesses, typical on-site support costs as well as replacing system cost becomes high. Consumers face even greater difficulties, since many lack technical expertise, and may not have immediate access to experienced troubleshooting or repair services. For both businesses and consumers, the cost of resolving a compatibility or reliability issue can quickly escalate beyond the original price of the Tablet. Intel's comprehensive validation program plays a critical role in helping to keep these costs down, reducing customer risk, and increasing the long-term value of newly purchased Tablet.

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

2.2.1 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 SoC and its' different IP like CPU,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.

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 SoC 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.

2.2.2 Stage 2: Analog Validation

As Tablet PC market growing very fast, performance demands continue to climb, the electrical integrity of SoC and other IPs 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: 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 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 SoC is validated under real world scenarios.

2.2.3 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 different boards and add-in cards, number of peripherals, and so many applications. OS testing includes multiple versions of Microsoft Windows versions. 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 Fiber Channel. 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.

2.2.4 Stage 4: Software Validation

Intel develops all core software components for its Tablet 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.

2.2.5 Stage 5: Platform Validation

This is a final phase in validation cycle, in this stage all the subsystems like communication module, graphics, power management IC, sensors 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.

2.3 Platform Validation Challenges

One of the most difficult decisions a platform design engineer has to make is when a design is ready to go to High Volume Manufacturing production. What level of platform testing, platform verification and platform validations required to achieve the minimum confidence level that is required for HVM? This decision process is different for every project and is subject to the level of confidence and acceptance of risk required by the program leadership.

2.4 Platform Validation working Areas

Following picture depicts the test areas that will be under the scope of platform validation. This covers feature functional validation, platform features and platform level use case scenarios.

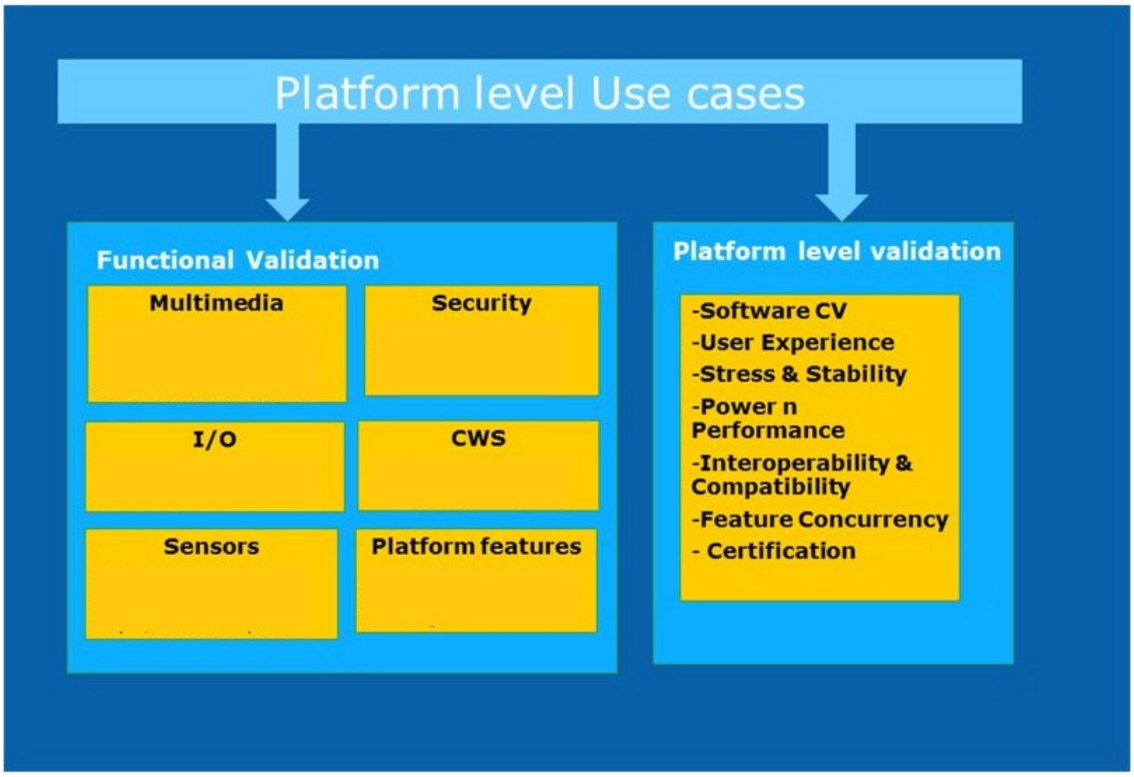


Figure 2.1: Platform Validation working Areas

- **Functional:** Functional validation covers the full feature validation of all the platform features.For example,Validation of Communication module,sensors etc.

- **Stress and stability:** Under stress and stability testing,validate functionality for extended hours of usage. Stress testing will be done for every interface of the platform.

- **Performance:** Monitoring the system performance like CPU Usage, Time taken for upload/ download, Data transfer from various storage devices, responsiveness of the system from sleep, idle states, touch responsiveness, battery life are measured for key use cases.

- **Power Management:** Validation of platform and OS Power management features like CPU state Residency,System state Residency, Display self-refresh.

- **Compatibility:** Compatibility testing will be performed to verify proper functionality of multiple vendors for the external peripherals like different makes of SD Cards, different makes of USB devices, HDMI monitors etc

- **User Experience/ Software Compatibility Validation:** Subjective validation of system performance, responsiveness, time taken to open top 20 web sites, 2D, 3D Games, third party applications will be validated as part of User experience and Software CV. These validation cycles will be performed once in a milestone, post Alpha.

- **Certification:** Windows Hardware Quality Labs (WHQL) is a Microsoft procedure for certifying that the hardware for peripherals and other components is compatible (works as expected) with Microsoft Windows operating systems. WHQL provides test kits to third-party developers so that they can test their product's compatibility. Products that are submitted to and meet the tests at Microsoft are allowed to display the Microsoft Windows logo on their marketing materials and are included in Microsoft's Hardware Compatibility List.

2.5 Scope of System BIOS on Platform Validation

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

2.5.1 System BIOS

The basic input/output system (BIOS), also known as the System BIOS , is a de facto standard defining a firmware interface. The BIOS software is built into the Tablet PC, and is the first code run by a Tablet 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 Tablet 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, memory and other hardware. The BIOS then locates software held on a peripheral device (designated as a 'boot device'), such as eMMC or a USB , and loads and executes that software, giving it control of the Tablet 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 . Typically this is a menu system accessed by pressing a certain key on the keyboard when the PC starts. In the BIOS user interface, 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 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 and Linux kernels 2.6.1 and greater builds (and in Mac OS X on Intel-based Macs).

2.6 Advance Configuration and Power Management

The Advanced Configuration and Power Management Interface (ACPI) specification 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 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,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.

Chapter 3

Tool features and Definition

3.1 Tool features

This tool is design to measure residency of CPU's power management state residency,CPU's performance State residency and System active sleep state residency.

3.2 C-State Definition

North Complex has primary control for dynamically changing processor C states. The Dynamic C-state control is handled using the following signals:

- **STPCLK#:** Used to halt CPU instruction stream, turns off gated clock in core 0 CPU
- **CPUSLP#:** Turns off free running clocks in core CPU
- **DPSLP#:** Turns core PLL off CPU
- **DPRSTP#:** Used to Mux the VID values in C4-C6 CPU
- **VIDEN:** Used to latch VID values for C6 MSIC

3.2.1 C0 State - Full On

This is the only state that runs software. All clocks are running. STPCLK is deasserted and the processor core is active. The processor services snoops and maintain cache coherency in this state. All power management for interfaces, clock gating, etc. are controlled at the unit level

3.2.2 C1 State - Auto-Halt

The first level of power reduction occurs when the processor executes an Auto-Halt instruction. This stops the execution of the instruction stream and greatly reduces the processors power consumption. The processor can service snoops and maintain cache coherency in this state. North Complex does not distinguish C1 from C0 explicitly

3.2.3 C2 State - Stop Grant

The next level of power reduction occurs when the processor is placed into the Stop Grant state by the assertion of STPCLK. The processor can service snoops and maintain cache coherency in this state. North Complex only supports receiving a single Stop Grant. It is expected that multi-core processors collect all thread's stop grants and issue a single stop grant to the North Complex.

Key features:

- Disable any FSB logic when not servicing snoop requests
- Master and Slave portions of the DLLs may be disabled when no read access is pending, the internal graphics controller is completely idle, and the display requirements are within a predefined limit.
- Memory in self-refresh when no access is pending.

Entry into the C2 state will occur after the CPU requests a C2 (or deeper) transition. C2 state will be exited, entering the C0 state, when a break event is detected. North

Complex must ensure the DLLs are awake and the memory will be out of self-refresh at this point. Upon STPCLK assertion, the CPU is not guaranteed to recognize transitions or take the proper behavior on special control and interrupt pins. Therefore, North Complex will freeze the state of these pins while STPCLK is asserted, and not change the values until CPU has returned to C0.

3.2.4 C3 State - Deep Sleep

In this state the CPU shuts down its PLL and cannot handle snoop requests. During the C3 state, the North Complex will continue to handle traffic to memory so long as this traffic does not require a snoop (i.e. no coherent traffic requests serviced). The features of this state are the same as C2 except that all snoop requests initiate the deassertion of CPUSLP such that the FSB can be snooped. Entry into the C3 state will occur after the C2toC3 timer has expired. North Complex ensures all requirements for C3 are met before asserting the signal sequence. The exit from C3 occurs when the North Complex detects a snoopable event or a break event, which would cause it to wake up the CPU and initiate the C0 sequence.

3.2.5 C4 State - Deeper Sleep

This state allows some further power optimization beyond the C3 state. The DPRSTP signal is asserted to the CPU to mux out the VID values for power saving. All functionality described above for C3 are applicable for C4, except that it is entered by receiving a C4 request by the CPU/OS

3.2.6 C5 State - Deeper Sleep

Is a case where CPU's resulted C-State is C4 while its L2 has been flushed, thus the Punit no longer need to wake the CPU for C4 popup snoops as the CPU doesn't contain any dirty data.

3.2.7 C6 State

Prior to entering C6, the CPU will flush its cache and save its core context to a special on-die SRAM on a different power plane. Once the C6 entry sequence has completed, the CPU's core voltage can be completely shut off. The key difference for the North Complex between C4 and C6 is that since the CPU's cache is empty, there is no need to perform snoops on the FSB. This means that bus master events (which would cause popup from C3/4 to C2) can be allowed to flow unimpeded during C6. However, the CPU must still be returned to C0 in order to service interrupts pin. A residency counter, C6C, in North Complex is read by the CPU to enable an intelligent promotion/demotion based on energy awareness of transitions and history of residencies/transitions.

3.2.8 C-state Residency Counters

Deeper C-states such as C5 and C6 have high energy cost for transitions. However, ACPI does not have a concept of energy cost associated with the transitions on C-states. NORTH COMPLEX keeps track of time of residency in deep C-states via the C6C register in the ACPI PBLK, since the CPU loses its internal clocks and voltage in deep C-states. The counter is 27 bits, which provides 227us= 2.23 minutes, and tracks the last entered C state.

3.3 P-state definition

Processor performance states (Px states) are power consumption and capability states within the active/executing states. P-states provide a way to scale the frequency and voltage at which the processor runs so as to reduce the power consumption of the CPU. The number of available P-states can be different for each model of CPU, even those from the same family. The Px states are briefly defined below:

3.3.1 P0 Performance State :

While a device or processor is in this state, it uses its maximum performance capability and may consume maximum power.

3.3.2 P1 Performance State :

In this performance power state, the performance capability of a device or processor is limited below its maximum and consumes less than maximum power.

3.3.3 Pn Performance State :

In this performance state, the performance capability of a device or processor is at its minimum level and consumes minimal power while remaining in an active state. State n is a maximum number and is processor or device dependent. Processors and devices may define support for an arbitrary number of performance states not to exceed 16.

3.4 S-state definition

3.4.1 S0-state

This state called as System Active. Component of SoC are Active and CPU may be transition freely in and out of C-states.Intel introduce new states for the further optimization of power in system active state which is called as S0ix states.

3.4.2 S0i1-state

This state called as "AOAC Stand-by" or "Active Stand-by". Used during idle periods when user is interactively using device. CPU in C6 retained.

- Home screen, web browsing, email

3.4.3 S0i2-state

This state called as "Active Stand-by" or "Sleep". Used during extended idle periods when the user is passively using the device. CPU in C6.Data retained in shared SRAM.

- Music playback, voice call

3.4.4 S0i3-state

This state called as "Deep Sleep". Used during extended idle periods when the user is actively using the device. CPU in C6.Data retained in shared SRAM.

- Sleep mode, always connected
- Able to wake from user or platform

Chapter 4

Implementation

4.1 Problem statement

Today, Complexity of platform has increased with incorporating SoC and Dozens of Platform component and supporting of multiple Operating systems. Validation of SoC and platform component is immensely complicated. One of the most important features of Platform is Power Management and Performance of system. The aim of this project to develop a tool which use to measure attributes contributing to Power and Performance (PnP) on SoC and other peripheral devices on 32nm based SoC Tablet Platform with supporting Windows 8 OS.The features supported by this tool is C-state Residency, P-State Residency, S-State Residency etc.

4.2 Basic Architecture

This tool is use for measuring the power and performance of different platform with supporting Windows 8 OS validation perspective. Basically Tool Architecture is comprise of Two parts:

- Framework
- Driver development

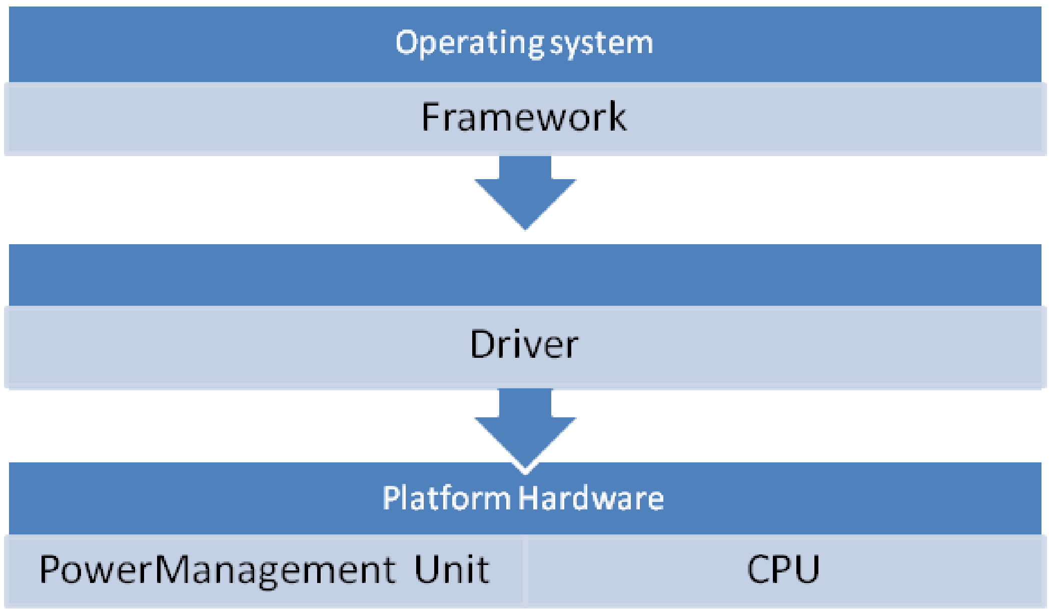


Figure 4.1: Basic Architecture

4.2.1 Framework

Framework is an application that work at user layer. Framework is responsible for setting up the parameter and preparing the system for capture of data. The quintessence of this framework is its consistency across the platform. This also defines the consistency of data across the platform and across the operating systems across various configurations. Framework is responsible for

- Process and consolidate user input for the capture of data
- Synchronize with driver
- Post processes the captured data for reporting.

4.2.2 Driver

Drivers are responsible following:

- Configuring the system for Capture
- Capturing the data from relevant source on the platform
- Timely polling

4.3 Platform Hardware requirement

Platform Hardware is consist of SoC (includes CPU, Power management unit etc.) and platform power management IC, sensors, LPDDR DRAM, Communication Module etc. Our focus is Power management Unit for measuring CPU PnP state transition and System Active states transition.

4.3.1 Overview of Power Management Unit

The Power Management unit controls all CPU power state transitions, and S0ix. The Power management unit interfaces all other units via the message bus, IO space via the backbone, and memory through a dedicated DMA interface. Most of this functionality is programmed into an embedded 8051 microcontroller, with external controls and RAM implemented in support logic. For the most part, every function of the P unit is a programmed subroutine in the microcontroller's "ROM".

4.4 Driver development

Tool drivers are components that enable the operating system (OS) and user applications to interact with peripheral hardware that is integrated or attached to a target device, such as accessing MSR register for our requirement ,the Peripheral Component Interconnect bus, keyboard, mouse, serial ports, display, network adapter and storage devices. Rather than accessing the hardware directly, the operating system loads the corresponding device drivers, and then uses the functions and input/output (I/O) services that these drivers provide to carry out actions on the device.

4.4.1 WDM Architecture

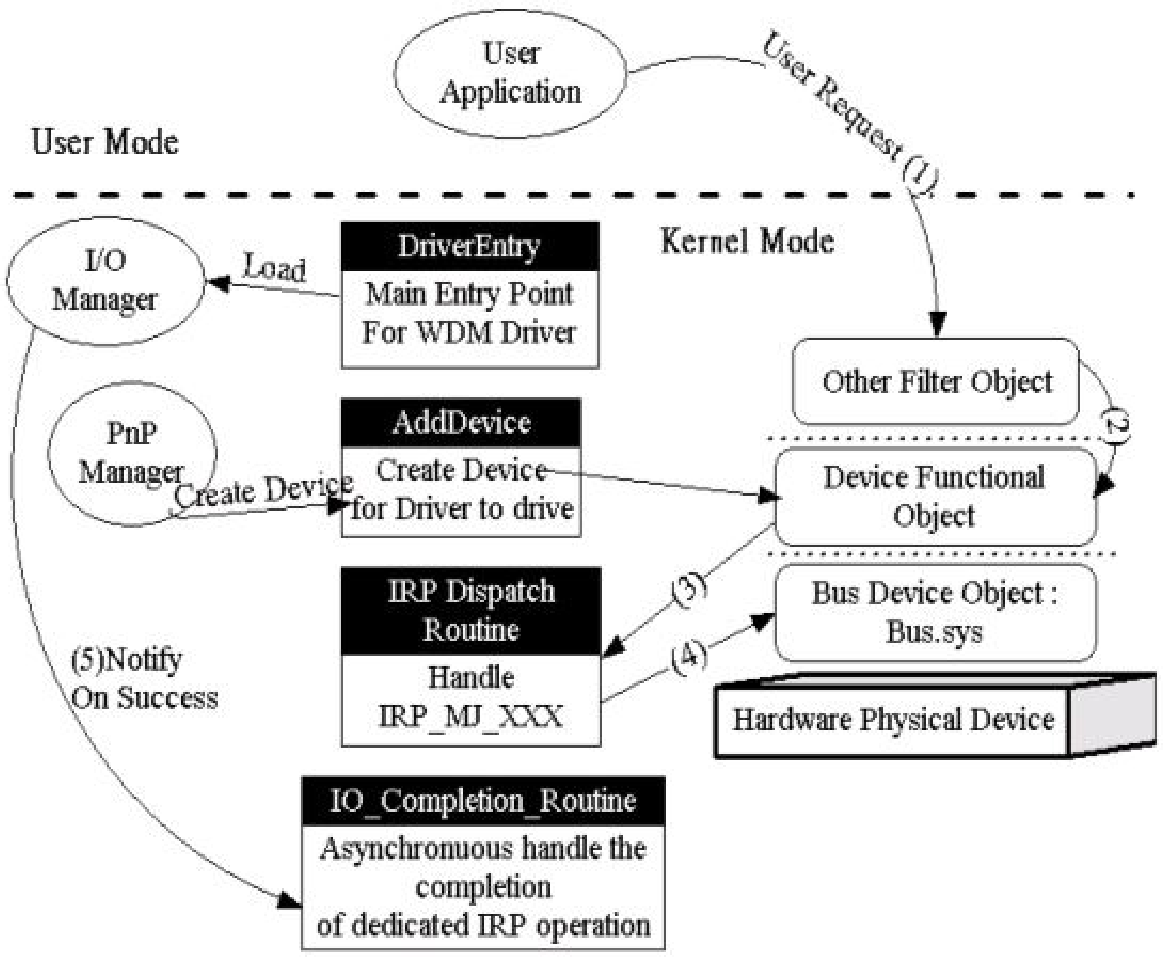


Figure 4.2: WDM Architecture

WDM drivers are layered in a complex hierarchy and communicate with each other via I/O request packets.WDM driver architecture is used for developing driver.There are mainly four hierarchy structure used for developing driver which is shown in the above figure.

- **DriverEntry routine :**The DRIVER_OBJECT is a data structure used to represent this driver. DriverEntry routine will use it to populate it with other entry points to the driver for handling specific I/O requests. This object also has a pointer to a DEVICE_OBJECT which is a data structure which represents a particular device. A single driver may actually advertise itself as handling multiple devices, and as such, the DRIVER_OBJECT maintains a linked list pointer to all the devices this particular driver services request for. We will simply be creating one device.

The next part is to actually put things in the DriverEntry routine. The first thing we will do is create the device. This is generally because a driver is usually associated with hardware but this is not the case. There are a variety of different types of drivers which operate at different levels, not all drivers work or interface directly with hardware. Generally, you maintain a stack of drivers each with a specific job to do. The highest level driver is the one that communicates with user mode, and the lowest level drivers generally just talk to other drivers and hardware. There are network drivers, display drivers, file system drivers, etc., and each has their own stack of drivers. Each place in the stack breaks up a request into a more generic or simpler request for the lower level driver to service. The highest level drivers are the ones which communicate themselves to user mode, and unless they are a special device with a particular framework (like display drivers), they can behave generally the same as other drivers just as they implement different types of operations

- **IRP Routine :** The data structure that encapsulates the IRP not only describes an I/O request but also maintains information about the status of the request as it passes through the drivers that handle it. Because the data structure serves two purposes, an IRP can be defined as:

  - a container for an I/O request: The operating system presents most I/O requests to drivers using IRPs. IRPs are appropriate for this purpose because processed asynchronously,can be canceled,are designed for I/O that involves more than one driver.

    Following the IRP header is an array of sub-requests. An IRP can have more than one sub-request because IRPs are usually handled by a stack of drivers. Each IRP is allocated with a fixed number of such sub-requests, usually one for each driver in the device stack. This number typically matches the StackSize field of the top device object in the stack, though a driver in the middle of a stack could allocate fewer. If a driver must forward a request to a different device stack, it must allocate a new IRP.

Each sub-request is represented as an I/O stack location (a structure of type IO_STACK_LOCATION), and the IRP typically contains one such I/O stack location for each driver in the device stack to which the IRP is sent. A field in the IRP header identifies the I/O stack location that is currently in use. The value of this field is called the IRP stack pointer or the current stack location.

- **IO completion routine :** A driver should return STATUS_PENDING from a dispatch routine when it cannot complete an I/O request synchronously in a timely manner. Understanding when to return STATUS_PENDING is a problem for many driver writers.

  A driver must return STATUS_PENDING if:

  - Its dispatch routine for an IRP might return before the IRP is completed.
  - It completes the IRP on another thread.
  - The dispatch routine cannot determine the IRP's completion status before it returns.

4.5 Implementation of Framework for User mode Application

Driver contains address of MSR register for example C-states, P-states, S-states and performs the transaction based on the IRP. IOCTL will help to read the value from hardware register or write the value to the register from the user interface.so that driver capture data dynamically based on the given interval time by user.IOCTL and MSR will cover in these section later.

User framework develop using VC++ using windows console application to communicate the driver. These frame work consist of manipulation of the IRP data and perform the calculation part of final result. Also It will include the software timer to set the time interval and logic used for generating reports.

4.5.1 IOCTL

The IOCTL is used as more of a communication between the driver and application rather than simply reading or writing data. Generally, the driver exports a number of IOCTLs and defines data structures that would be used in this communication. Generally, data structures should not contain pointers since the I/O Manager cannot interpret these structures. All data should be contained in the same block. If you want to create pointers, you can do things such as create offsets into the block of data past the end of the static data so the driver can easily find this information. If you do remember however, the driver does have the ability to read user mode data as long as it's in the context of the process. So, it is possible to implement pointers to memory and the driver would need to copy the pages or lock the pages in memory (implement basically buffered or direct I/O from within the driver itself, which can be done). The user mode process will use the ”DeviceIoControl” API to perform this communication.

The first thing we need to do is define the IOCTL code to be used between the application and the driver. First, to relate the IOCTL to something in user mode, you may think of it as a Windows Message. It's simply a value used by the driver to implement some requested function with predefined input and output values. There is a little more to this value than a Windows Message however. The IOCTL defines the access required in order to issue the IOCTL as well as the method to be used when transferring the data between the driver and the application.

4.5.2 MSR

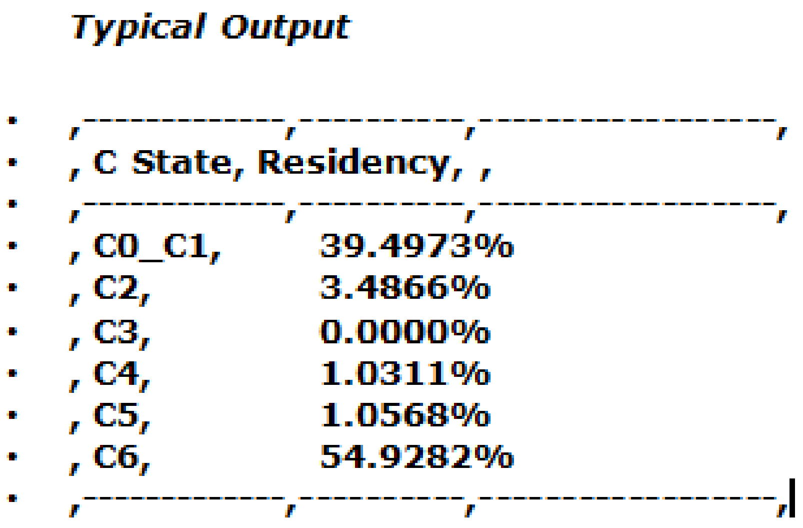
- Model-specific registers (MSRs) are control registers provided by processor implementations
- To provide system software with features that are provided on specific processor implementations, but not others.
- MSRs are used for performance monitoring, debugging, testability and program execution tracing, but also to enable and disable certain model-specific features of the processor implementation.

4.6 Calculation and Results

4.6.1 C-state residency

For example: C6 5 ticks = 5usec

- (5usec/10usec)*100=50

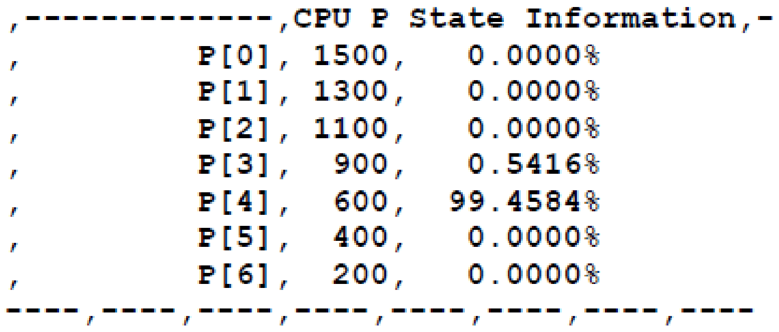


Typical Output

```
•  ,------------,----------,-----------------,
•  , C State, Residency, ,
•  ,------------,----------,-----------------,
•  , C0_C1,      39.4973%
•  , C2,         3.4866%
•  , C3,         0.0000%
•  , C4,         1.0311%
•  , C5,         1.0568%
•  , C6,         54.9282%
•  ,------------,----------,-----------------,
```

Figure 4.3: C-state residency

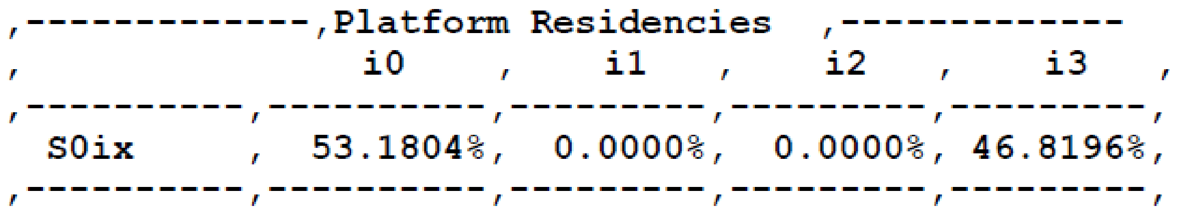
4.6.2 P-state residency



```
,-------------,CPU P State Information,-
,         P[0], 1500,   0.0000%
,         P[1], 1300,   0.0000%
,         P[2], 1100,   0.0000%
,         P[3],  900,   0.5416%
,         P[4],  600,  99.4584%
,         P[5],  400,   0.0000%
,         P[6],  200,   0.0000%
----,----,----,----,----,----,----,----
```

Figure 4.4: P-state residency

4.6.3 S0ix-state residency



```
,-------------,Platform Residencies   ,-------------
,               i0     ,     i1    ,     i2    ,     i3    ,
,----------,----------,---------,---------,---------,
  S0ix     ,  53.1804%,  0.0000%,  0.0000%, 46.8196%,
,----------,----------,---------,---------,---------,
```

Figure 4.5: S0ix-state residency

Chapter 5

Conclusion

By using this framework, we can validate,

Transition of different CPU state and based on that can decide Residency of different c-state during given time period and expected behavior of CPU power consumption.

Transition of different CPU frequency and based on that can decide P-state residency and expected behavior of CPU Performance.

Transitions of Different System Active Sleep states and based on decide S0ix residency and expected behavior of system state power consumption.

By using these tool,measure attributes contributing to Power and Performance (PnP) on SoC and other peripheral devices on 32nm based SoC Tablet Platform with supporting Windows 8 Operating System

References

[1] Platform Architecture Document, Intel

[2] Power Management Hardware Archiecture Specification, Intel

[3] BIOS writer Guide,intel

[4] MSDN library,Microsoft