Pentium® III Processor Serial Number Feature and Applications

Stephen Fischer, BMD-FM Design, Intel Corp.
James Mi and Albert Teng, Content Group, Intel Corp.

Index words: Pentium® III, Internet, Java*, asset management, information management

ABSTRACT

With the ever-growing importance of the Internet in the everyday life of an individual or a business user of a personal computer, the ability to have some form of unique identifier for that computer has become increasingly important. Applications ranging from system management for reducing Total Cost of Ownership (TCO) and electronic commerce to information management can expect to benefit from such a capability.

In response to this need, the Pentium® III processor has incorporated a serial number capability into the existing instruction set. The serial number feature makes use of information programmed onto the die during manufacturing, designed to create a unique number that is readable by external software.

Intel concerns about user privacy led to the incorporation of a user hardware disable feature for the processor serial number.

INTRODUCTION

The Intel® processor serial number (which for brevity will be referred to as ps#) refers to a new feature introduced with the Pentium® III processor, namely a unique numeric identifier. This serial number can be read by external software.

With the availability of a processor-based serial number, new classes of software applications are more easily enabled. Moreover, electronic transactions via the Internet can be more easily enabled by using the ps# as an added level of support for authentication. Corporations can make use of the ps# to facilitate system configuration and tracking, thereby improving manageability. Sensitive data can be closely controlled by binding the ps# information to the accessibility of the data.

This paper defines the processor serial number feature and explains how it is implemented on the Pentium III processor. We also describe some of the applications that are enabled with this capability, and give an overview of an application framework that provides CPU identification, based on the ps# in an open network environment, and limits cross-correlation of information across Web sites.

ARCHITECTURE AND IMPLEMENTATION

The ps# capability introduced in the Pentium® III processor is communicated through an extension of the existing CPUID instruction [1]. The CPUID instruction is responsible for returning specific parameters of the processor to external software. The type of parameters include items such as the product family, model, and stepping, as well as feature-specific attribute information such as a feature flags field for indicating what functions are available for this processor. Including the processor serial number information in the list of possible parameters that can be returned was therefore a natural extension of the CPUID instruction. Since this instruction can be executed at all privilege levels (PL0 – PL3), it is available for execution at the application level as well as the OS level, an important distinction that enables a much wider range of uses of the ps# feature.

Parameter information such as the stepping number or feature flag bits are returned in the general purpose registers EAX, EBX, ECX, and EDX when the CPUID instruction is executed with a specified input index value held in EAX.

```
Case EAX=0;
{
       EAX           = maximum index supported
       EBX:EDX:ECX   = "GenuineIntel"
}
Case EAX=1;
{
       EAX           = Family:Model:Stepping
       EBX:ECX       = reserved
       EDX           = feature flags (New ps# feature flag bit 18)
}
Case EAX=2;
{
       EAX:EBX:ECX:EDX  = cache and TLB parameters
}
Case EAX=3;
{
       EAX:EBX       = reserved
       ECX:EDX       = processor serial number  data
}
```

Figure 1: CPUID instruction definition

The ps# information is returned by the addition of a new index (EAX=3). The presence of the ps# feature is also indicated by the setting of a new feature flag, bit 18. This allows external software to determine if the ps# feature is supported and enabled in the processor. Figure 1 summarizes the definition of the CPUID instruction supported by the Pentium III processor.

To address potential concerns about compromising user privacy through the visibility of the processor serial number, a capability is incorporated that allows a user to choose whether to enable or disable this feature. This capability is intended to be under the user or system manager's control. This is implemented through the addition of a new read/write control register disable bit with a "sticky" property. During execution of the CPUID instruction, the internal microcode of the processor polls this bit to determine if ps# information should be reflected back to the CPUID instruction-level functionality. Once the bit is set to a '1' (ps# disable), it cannot be cleared back to a '0' through software means; only a hardware reset of the processor can clear the disable bit, thus preventing subsequent software from overriding the user preference setting after a potential software disable action has been performed. It should also be noted that the ps# disable bit is accessible only at the highest privilege level (PL0). This level of accessibility keeps the user or system manager preference setting decision with supervisory or initialization software such as the system BIOS.

The ps# information returned by the Pentium III processor is derived from on-die polysilicon fuse bits programmed at wafer sort. The underlying microcode supporting the CPUID instructions reads the logical programmed values of these internal fuse bits and concatenates them to form a 64-bit value returned in the general purpose registers EDX and ECX.

The underlying fuse technology is based on a novel silicon approach that uses a Ti-silicide layer on top of a polysilicon line [2]. Programming occurs by a current stress that is high enough to cause agglomeration of the Ti-silicide. A current mirror sensing circuit is used to measure the programmed fuse resistance relative to an unprogrammed reference fuse and return a logical value. The technology has yielded near 100% programming success and maintains this reliability under thermo-mechanical and bias-temperature stress conditions. Redundant fuse elements for each logical fuse bit are incorporated to further increase the reliability for a successful programmed value, yielding a robust process for deriving and programming the serialized value for the ps#, in manufacturing of the Pentium III processor.

APPLICATION USAGE MODELS

Example 1: Improving Manageability, Reducing TCO

In large enterprise environments, IT managers face daily challenges to ensure a well-managed and smoothly running computing infrastructure. The Intel® Pentium® III processor and its ps# give IT departments a new tool to improve manageability and lower the total cost of ownership of PCs.

In the past, IT departments utilized a variety of methods, including user name, MAC address, IP address, and GUID to identify hardware. However, none of these methods are as consistent and reliable as the ps#, which cannot be erased or changed. With a ps#, it is easy to identify a specific PC, even if the system changes users, network cards are swapped out, or the system software and BIOS are reloaded. The ps# also makes it possible to report more reliably on software asset management: IT managers can know with a higher level of certainty which software is running on each system and who the users of the software program are. It can also assist Help Desk personnel in troubleshooting problems even when a PC's hard disk has crashed.

For system configuration and software updates, companies can use the ps# as a way of reliably identifying PCs pre-boot and post installation remotely. If support technicians know the processor serial number ahead of time, they can enter the number in the database and pre-program the software to be delivered when the PC is placed on the network. This reduces on-site engineering visits and automates the configuration process, saving time and reducing support expenses.

When a processor fails in a multiprocessor or clustering environment, it is difficult to determine which specific processor failed. With the Windows NT* operating system, the logical processor can be identified but not mapped to the physical processor. The ps#, however, allows IT staff to determine the exact point of failure, thereby enabling them to route work around the problem processor. This can significantly improve load balancing and fault tolerance, and it can increase the system's availability to the user.

Example 2: Enhancing Management for E-Business

Internet-based Electronic-Business (E-Business) gives companies new freedom to push and pull information to and from one another, but also increases the need to ensure that the information reaches only its intended recipients. The ps# can be invaluable in this regard.

Using present technology, individuals and businesses can authenticate who is accessing the information on their personal computers and their company network by combining any two or three variables: the traditional *something you know* mechanisms such as login names and passwords; *something you have* items such as hardware keys (dongles) and smartcards, and *something you are* aspects such as biometric measures.

With the launch of the Intel Pentium III processor and its ps# technology, the PC now has another *something you have* item. It is an access token that can be used in conjunction with passwords to help ensure that only the intended platform receives sensitive corporate information. For example, an Internet-based travel agency network can validate a system's processor serial number to make sure that sensitive pricing information is pushed only to authorized travel agents' machines. The increased identification offered by the ps# also helps corporate intranets extend information to employee desktops, offering employees greater real-time access to their 401(k) plans, payroll, and other personal data once their ps# is validated. The ps# also allows businesses to broadcast sensitive video with synchronized presentations by adding another layer of authentication prior to pushing the presentation out to the user.

In business-to-business transactions, corporations can bind the ps# to their digital certificates and internal or external certificate authorities. Business partners can then gain access to private information only if they have their corporate certificate and are accessing the data from a validated platform. (For more information on how to validate system identity, see the section entitled "System Verification Based on Processor Serial Number" in this paper.)

Example 3: Information Management

As the flood of information rises and the PC becomes the primary vehicle for processing, storing, and accessing information, the management of information poses a greater challenge. The ps# provides a non-intrusive identifier that enables information service providers to customize the data and services that are delivered to the end user. The ps# also provides a better way to track and protect important or sensitive information, and it can improve applications such as data backup and restore protection, removable storage data protection, managed access to files, and confirmation of document exchange between appropriate users.

SYSTEM VERIFICATION

It is a challenging task to design a software system that can reliably identify a system in an open network environment, based solely on the serial number of a processor. Because the client system could be a hostile system controlled by a potential hacker, it is difficult for the server to determine whether the returned ps# information from the client system is valid or spurious. The Processor Serial Number Verification Reference Implementation (RI) offers a basis to solve this problem. The RI was a joint effort by Intel and Independent Software Vendors (ISV) to provide a way to extract the ps# from a client system in an open network environment

and limit cross-correlation of information across Web sites.

Tamper Resistance

The RI's software agents are downloaded over an open network and are therefore exposed to attacks by hackers. To protect against these attacks, the software agents are designed using special tamper resistant techniques, appropriately called Tamper Resistant Software (TRS) agents. The TRS agents automatically detect and protect against potential hacker attacks. Typically non-processor-based hardware authentication solutions imply the use of privileged instructions; it is not available at the application level directly. The additional software layers such as device drivers are exposed to more hacker attacks. Processor serial number instruction can be executed at the application level directly and does not require special drivers. It makes the tamper resistant protection stronger. These TRS agents are available from different ISVs. A common set of API functions has been defined and made available to these vendors so that developers can easily use agents from a number of vendors interchangeably.

To provide safety against impostors, the framework adopts a protocol whereby the authentication or verification of the client happens on the server. Agents are used once for a short time and then are discarded, thus enhancing protection.

Privacy Protection

Authentication mechanisms play an important role in Web-based applications. However, some users or businesses may not honor a consumer's right to privacy, and they gather personal information about the consumer without the consumer's consent. Intel has taken several measures to address consumers' privacy concerns not only in the design of the processor serial number feature, but also in providing certain utility tools. Several RI features work to further enhance the protection of a user's data:

- Software agents that gather the ps# are packaged in a digital container (a cabinet file for Internet Explorer*, and a Jar file for Netscape Navigator*) that is then digitally signed by the Web service provider and delivered to the client system. When the Web browser sees the container, it prompts the user to grant access rights to the software, ensuring that the ps# cannot be collected without the user's consent.
- The ps#, once read, is transformed into another unique identifier by hashing it with a service ID. The hashed value is then sent by the client agent to the server to be stored in the user database.
- The service ID is unique to each service provider. This precludes different Web sites from correlating user profiles due to the non-communicative characteristic of the hashing algorithm.

The hashing algorithm is designed to be a one-way, collision-free algorithm, which means one cannot infer the processor serial number given the hashed value and the service ID. Intel also recommends that Web service providers make their privacy policies available to the consumer.

Performance Considerations

Another important attribute of the RI design is the short download time for the agents. If the size of the software agents is large, the user might have to wait for a long time before getting access. To reduce download times, agents used in the RI are limited to about 35 Kbytes. However, the quality of protection is proportional to the size of agents: the larger the size, the better the protection provided. A balance was reached with a small agent that can protect against attacks for a sufficient time. Protection is augmented by dynamically renewing the agents and by using a time-out mechanism on the server.

REFERENCE IMPLEMENTATION ARCHITECTURE

The RI framework consists of a client module, a server module, and a protocol for communication between the client and the server.

The client module consists of two types of client agents: a registration agent, which is a non-armored module for client registration; and a verification agent, which is a TRS armored module for client verification. Each agent consists of a Java* applet and native code DLL that are packaged together in digitally signed containers.

The server module manages client sessions and authenticates the client system in addition to providing access to the Web site. The Web server also stores the registration code with the user name and password in a backend database.

*Other brands and names are the property of their respective owners.

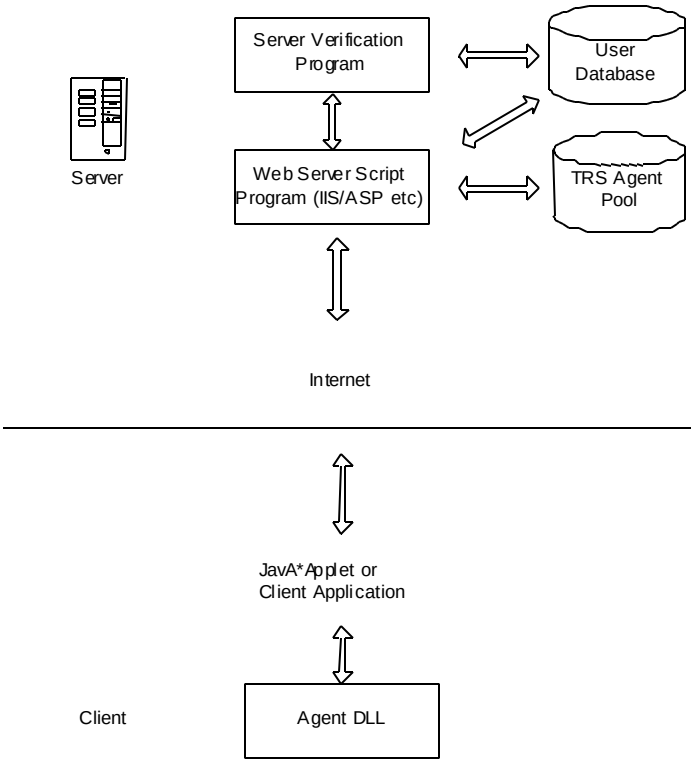

Figure 2: Reference implementation architecture

System Verification Flow

When a user first logs on to the Web server, the server asks for a user ID and password and then downloads a registration agent to the client system. The registration agent computes a registration code that is a hash of the ps# and service ID, and then returns it back to the server to be stored in the user database with the user's name and password. The server then sends a randomly selected verification agent from a pool of agents. Each agent is tamper resistant and embeds a unique secret value. It is only used once during a pre-defined time duration (usually about ten minutes). The agent is designed to sustain an attack by a very experienced hacker during that time period. Once the verification agent is downloaded to the client system, it calculates the verification code and sends the verification code to the server. As a measure of extra protection, the server times out during these sessions if it does not receive a valid verification code from the client within a pre-determined time.

The verification code is computed by first hashing the ps# and service ID (the same as the one used for registration). The resulting registration code is again hashed with additional unique secret values embedded within the verification agent. This results in a verification code, which is sent back to the Web server for verification.

After the server receives a valid verification code from the client, it first stores the returned value temporarily. Then the server calculates an authentication code by hashing the previously stored registration code with the unique secret value embedded in the particular verification agent sent to the client. The server then compares the authentication code with the verification code. If the two values match, the client system is authenticated, and the user can access content or obtain the requested service.

Supported Environment

Client agents support Microsoft* Windows* platforms (Windows NT*, Windows* 98 and Windows 95*). The RI supports Microsoft Internet Explorer* 4.0, Netscape Navigator* 4.x, and AOL* browsers. Both uniprocessor and multiprocessor client systems are supported. For multiprocessor systems, the ps# is gathered consistently from the same processor selected from the set of available processors. Similar software can be developed to support other client operating systems, such as Linux.* For server environments, the Reference Implementation supports both Windows NT and Unix.*

CONCLUSION

The serial number feature of the Pentium® III processor is designed to provide a unique identifier for each processor shipped by Intel with this feature to be visibly retrieved using application software. The CPUID instruction enables a natural method for providing this information with minimal impact to the processor design or to future implementations.

As mentioned in this paper, several different categories of applications can greatly benefit from a processor-based serial number capability. The most obvious areas are enterprise asset management, information management, and management for E-Business.

Unlike other means of deriving unique identifiers, the ps# feature implementation of the Pentium III processor is not impacted by a change of system hardware or software configuration (i.e., network card, IP address, etc.). Embedding this feature in the processor provides multiple benefits:

*Other brands and names are the property of their respective owners.

- Consumers and service/content providers have greater confidence in this feature due to the increased tamper resistance of the unique identifier.
- Visibility of the ps# to application-level software. Typical non-processor based solutions imply the use of privileged instructions (PL0) not available to application-level code or common across platforms.

As a result, we expect the ps# feature of the Pentium III processor to be of particular value to groups such as information providers and IT managers, and also to consumers as new applications take advantage of this feature.

ACKNOWLEDGMENTS

We thank those who helped to initiate, define, and refine the processor serial number feature and application program. Among the most active were Rob Sullivan, Ticky Thakkar, Natasha Oza, Vishesh Parikh, Jim Kolotorous, Susan Wojcicki, and Peter Ruscito.

REFERENCES

[1] *Pentium® Pro Family Developer's Manual, Volume 2: Programmers Reference Manual*, Order Number 000900-001, Intel Literature Sales, Mt. Prospect, IL, 1996, pp. 11-73 to 11-79.

[2] Mohsen Alavi, Mark Bohr, Jeff Hicks, Martin Denham, Allen Cassens, Dave Douglas, Min-Chun Tsai, "A PROM Element Based on Salicide Agglomeration of Poly Fuses in a CMOS Logic Process," 1997 IEEE International Electron Devices Tech Digest, December 1997, pp. 855-858.

AUTHORS' BIOGRAPHIES

Stephen Fischer is a staff design engineer with Intel Corporation's Folsom Design Center, which is responsible for the microcode and microarchitecture related design of the Pentium® III processor. Prior to that, Stephen was involved in various programs including EISA chipset definition, PCI bus and chipset definition, and the Intel MMX™ technology definition. He received a B.S. degree in computer engineering from CSU-Sacramento in 1985 and currently holds six U.S. patents. His e-mail is sfischer@pcocd2.intel.com.

James Mi is manager of Enabling Technology with Intel's Content Group, which is responsible for application architecture and development. Prior to that, he worked in marketing, software and hardware development at Intel's Content Group, TCAD, and Flash TD. James received a B.S. degree in physics from Fudan University, China, in 1989 and an M.S. degree in EE from Princeton University in 1991. He joined Intel in 1992. He holds seven U.S. patents. His e-mail is james.mi@intel.com.

Albert Teng is director of New Technologies with a focus on client/server applications for enterprise/e-commerce solutions, security, and knowledge management. Previously he was the general manager of Intel China and held an engineering management position in the Microcomputer Labs. Before joining Intel in 1985, Albert worked at AT&T Bell Labs and at the Illinois Institute of Technology. He received his Ph.D. from Ohio State University in 1979. His e-mail is albert.y.teng@intel.com.