Intel® Extreme Tuning Utility, Version 6.0
BIOS Interface Specification

Revised for Skylake Support

Revision 0. 52

Last Update: June 20, 2014

Legal Notices and Disclaimers

INTEL CORPORATION MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. INTEL CORPORATION ASSUMES NO RESPONSIBILITY FOR ANY ERRORS THAT MAY APPEAR IN THIS DOCUMENT. INTEL CORPORATION MAKES NO COMMITMENT TO UPDATE NOR TO KEEP CURRENT THE INFORMATION CONTAINED IN THIS DOCUMENT.

THIS SPECIFICATION IS COPYRIGHTED BY AND SHALL REMAIN THE PROPERTY OF INTEL CORPORATION. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED HEREIN.

INTEL DISCLAIMS ALL LIABILITY, INCLUDING LIABILITY FOR INFRINGEMENT OF ANY PROPRIETARY RIGHTS, RELATING TO IMPLEMENTATION OF INFORMATION IN THIS SPECIFICATION. INTEL DOES NOT WARRANT OR REPRESENT THAT SUCH IMPLEMENTATIONS WILL NOT INFRINGE SUCH RIGHTS.

NO PART OF THIS DOCUMENT MAY BE COPIED OR REPRODUCED IN ANY FORM OR BY ANY MEANS WITHOUT PRIOR WRITTEN CONSENT OF INTEL CORPORATION.

INTEL CORPORATION RETAINS THE RIGHT TO MAKE CHANGES TO THESE SPECIFICATIONS AT ANY TIME, WITHOUT NOTICE.

Revision History

Revision	Date	Reason for Changes
0.50	September 6, 2013	Document branched from XTU BIOS Interface Specification Version 5.0. Initial draft for SKL specific revisions.
0.51	June 11, 2014	Updated tuning control IDs for 62h to 68h
0.52	June 20, 2014	Incorporated Ben's input

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 Purpose of this Document

The purpose of this document is to specify the BIOS interfaces necessary for implementation to support the Extreme Tuning Utility (XTU) application. It includes all the information necessary for someone to implement and use these interfaces. It will stand on its own and not be dependent on other documents to describe how to provide the BIOS interfaces.

1.2 Document Scope

This BIOS Interface Specification provides information regarding the programming model that is used for this module, any dependencies that exist within this module, and complete descriptions of the interfaces that are provided by this module. Let it be clear that the document provides only the interface, not the design or implementation of those interfaces.

1.3 Assumptions

Throughout this document technical terms regarding BYTEs, WORDs, DWORDs, and QWORDs are used. All references should be assumed to be little-endian. Also, BYTEs should be assumed to be 8 bits and WORDSs 16 bits.

1.4 Supported Platforms

The Intel® Extreme Tuning Utility supports a specific set of Intel microprocessor based platforms. XTU supports all Sandy Bridge, Ivy Bridge, Haswell and Broadwell based processors. This includes both Mobile and Desktop including the high-end desktop (HEDT) processors. It also includes processors with or without integrated graphics. This version will add support for Skylake mobile and desktop processors.

Acronym	Description	Comments
SNB	Sandy Bridge CPU and Cougar Point PCH (6 series chipset)	
SNB-E	Sandy Bridge based server platform	
IVB	Ivy Bridge (3rd generation) CPU and Panther Point PCH (7 series chipset)	
IVB-E	Ivy Bridge based server platform	
HSW	Haswell (4th generation) CPU and Lynx Point PCH (8 series chipset)	
HSW-E	Haswell based server platform	
BDW	Broadwell (5th generation) CPU and Wildcat Point PCH (9 series	

	chipset)	
BDW-E	Broadwell based server platform	
SKL	Skylake (6th generation) CPU and Sunrise Point PCH (10 series chipset)	

1.5 Terminology and Acronyms

Acronym	Description
ACPI	Advanced Configuration and Power Interface
ASL	ACPI Source Language
BCLK	Base Clock (aka Reference Clock) – The clock used as a source for many of the clock domains on the CPU and PCH
BIOS	Basic Input/Output System – This is the firmware responsible to boot a PC
CPU	Central Processing Unit – The main processor for a platform
HEDT	High-End Desktop
EAX	Register of the x86 processor
EBX	Register of the x86 processor
ECX	Register of the x86 processor
EDX	Register of the x86 processor
FIVR	Fully Integrated Voltage Regulator in the CPU package. This was introduced in HSW platform.
FW	Firmware
IA	Intel Architecture
ICC	Integrated Clock Controller
IO	Input/Output
IVR	Integrated Voltage Regulator
OS	Operating System
PCH	Platform Controller Hub
PCMCIA	Personal Computer Memory Card International Association – aka PC Card
PLL	Phased Locked Loop
RPM	Rotations Per Minute
SMI	System Management Interrupt
SPD	Serial Presence Detect – Non-volatile memory that is used on memory sticks to describe the characteristics of the memory

Acronym	Description
SW	Software
TDP	Total Design Power – The maximum power that a processor is designed to use
VR	Voltage Regulator – A circuit used to maintain a specific voltage in order to power another circuit
WDT	Watchdog Timer – A timer used to recover from a halted or hung platform state
XMP	Intel® Extreme Memory Profiles – Pre-defined Memory Overclocking Profiles defined as part of the SPD
XTU	Intel® Extreme Tuning Utility – Overclocking software provided by Intel

Table 1: Definition of Acronyms Used

1.6 Related Documents

Document Name	Revision	Doc Location
Advanced Configuration and Power Interface	3.0b	http://www.acpi.info/
Platform Performance Tuning Guide	SNB IVB HSW SNB-E IVB-E BDW HSW-E SKL	RS#29429 (Inactive) RS#30940 RS#32710 RS#30734 RS#32808 RS#33396 RS#33660 RS#33693
Extreme Memory Profile specification	2.0	CDI #541356

Table 2: Related Documentation

2 BIOS Interface Overview

The BIOS interfaces in the Extreme Tuning Utility serve two purposes for the XTU application. The first usage of this interface is to persist BIOS settings from the OS by providing mechanisms for reading from and writing to BIOS setup. The second purpose of the BIOS interface is to provide a mechanism that XTU can use to manipulate runtime, board-specific devices.

The XTU BIOS interfaces do not replace the need for the BIOS to implement core overclocking functionality. In order for the interfaces that are described within this document to function correctly, it is necessary for the BIOS to implement support for overclocking. This includes reference clock (or bclk) control, voltage control, manual memory timing manipulation, and more. Descriptions specific to each platform regarding implementation of core overclocking functionality is out of the scope of this document. Please refer to the appropriate Platform Performance Tuning Guide for direction in this area.

2.1 Key Concepts

There are two key concepts that should be understood by the BIOS engineer when implementing the XTU BIOS interface. The first of these concepts is the XTU ACPI Device and the purpose of this device. The XTU ACPI Device is generated by ASL that is written by the BIOS developer. The main purpose of this device is to provide a mechanism that can be used for passing platform specific information from the BIOS to the OS. It can optionally provide support for reading from and writing to platform specific hardware in runtime. The second main concept that is important to understand is the mechanism that XTU uses in order to persist data across reboots. In order persist BIOS setup information across reboots XTU passes updated information to the BIOS via an SMI. The associated SMI handler must interpret the data that is passed to the BIOS and store it in flash or another non-volatile medium where it can be integrated into future boots.

2.1.1 XTU ACPI Device

This device serves two main purposes. The first purpose is to pass the complete list of tuning controls (See ENUMERATIONS) which are supported by the platform along with the settings which they support to the XTU software. This listing of controls and settings allows XTU the ability to expose settings to the user which require a reboot. It also allows the XTU software the ability to expose platform specific hardware.

The second purpose of the XTU ACPI device is to allow for runtime control and monitoring of platform specific devices. The Run-Time Control Objects and Monitor-Only Objects described later in this document allow for both runtime control and monitoring style devices to be implemented.

2.1.2 XTU SMI Handler

The SMI Handler must be developed to allow for persisting data to the BIOS from the OS at run-time. A software SMI interface is used to pass control to the BIOS from the application. The SW command handler data is a piece of data that is sent to the application through the previously mentioned ACPI methods. Using this software System Management Interrupt (SMI) port and command data control will be passed to the BIOS for handling of some functionality. This functionality is for reading and writing BIOS setup data. This functionality is represented in Figure 2.

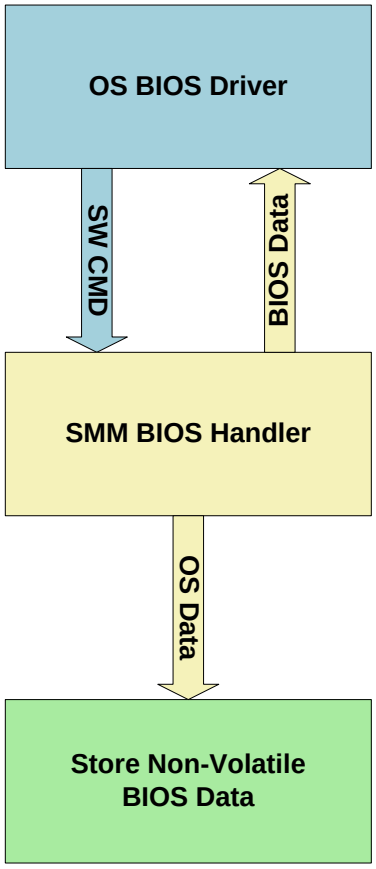
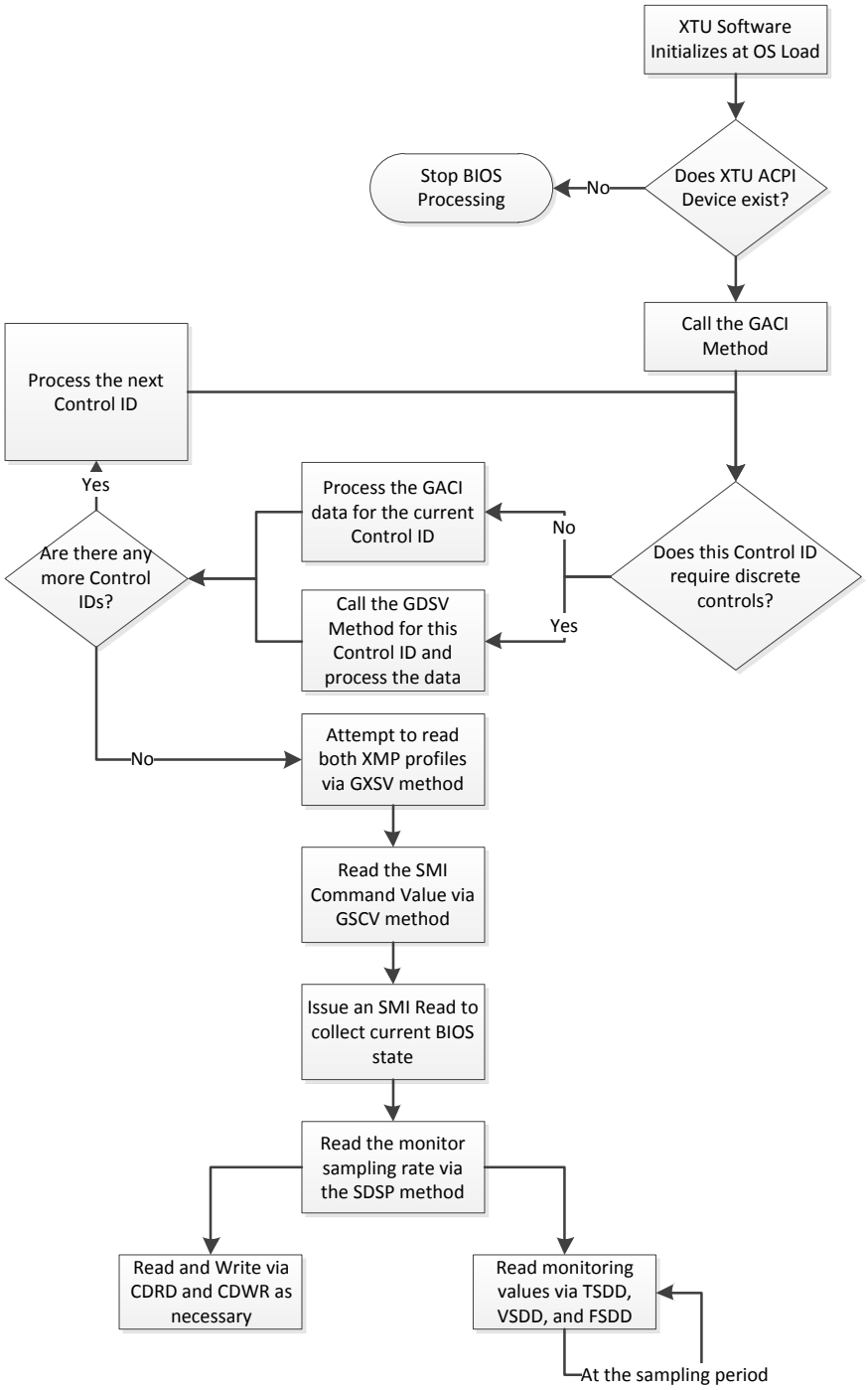


Figure 1: OS-to-BIOS Communications

2.2 Call Sequence

The following flow chart outlines the calls that will be made to BIOS by the XTU software. All calls referenced in this chart are defined in the INTERFACE DEFINITIONS section later in this document.



3 Interface Definitions

3.1 ACPI Device Interface

The custom XTU ACPI Device is the foundation of the new XTU BIOS Interface. This ACPI device definition is required in order to allow XTU to communicate to BIOS for the purpose of either persistence of setting values across reboots or control of run-time platform specific devices. The following sections will describe each of the required and optional structures, their purpose, and detailed descriptions.

All examples in the following section are referring to ASL code. For details on syntax and ASL conventions please refer to the Advanced Configuration and Power Interface Specification available at http://www.acpi.info.

3.1.1 Device Description

The following table provides details of the generic ACPI device. This device is what the XTU ACPI driver will register against and must be present in order to support either ACPI Control or Monitor-only methods and objects.

Identification Method	Value
_HID	INT3394
_CID	PNP0C02

Table 3: ACPI Device Identification

This device can be implemented under any scope of the platform ACPI's namespace, however, it is recommended to be implemented within the _SB scope.

Specifying the _CID ensures that the ACPI device does not show up in the Windows Device Manager as an "Unknown Device" with a yellow bang.

3.1.2 Object Overview

The following table provides an overview of the objects which are described in the upcoming sections. This provides a clear understanding of the various supported names and methods which make up the XTU ACPI device.

ACPI Object	Object Name	Type	Description
IVER	Version	Name	This object defines the version number of the interface.
GACI	Get Available Controls	Method	This object defines the Control IDs supported by the platform and includes the static information associated with those controls.
GDSV	Get Discrete Supported Values	Method	This object is used to describe a discrete set of display values when the control is unable to be described as a continuous set of values.

ACPI Object	Object Name	Type	Description
GSCV	Get SMI Command Value	Name	This object is used to describe the command that must be sent to the software SMI.
GXDV	Get XMP Display Values	Method	This object is used to retrieve the Control IDs and display values associated with a requested XMP Profile.
CDRD	Control Device Read	Method	This object is used to read the current state of a platform specific runtime control.
CDWR	Control Device Write	Method	This object is used to write the current state of a platform specific runtime control.
TSDD	Temperature Sensor Device Dump	Method	This object is used to get the current state of all temperature sensors on the system.
VSDD	Voltage Sensor Device Dump	Method	This object is used to get the current state of all voltage sensors on the system.
FSDD	Fan Sensor Device Dump	Method	This object is used to get the current state of all fan sensors on the system.
SDSP	Sensor Data Sampling Period	Method	This object is used to get the sampling period that should be used for all monitors.

Table 4: ACPI Device Object Overview

3.1.3 Generic Objects

The following objects provide the XTU software with standard information that describes the ACPI device interface.

3.1.3.1 Interface Version (IVER)

The IVER object evaluates to an integer that represents the version of this interface. It is a required object to be implemented on this interface.

The upper two bytes indicate the major version and the lower two bytes indicate the minor version.

```
Name (IVER, 0x00010000)     //Version 1.0
```

3.1.4 Control Detail Objects

The ACPI Control Detail Objects provide XTU with a variety of information about the platform. Specifically they provide information regarding which Control IDs are supported in runtime, which are persisted to the BIOS, and what settings are available for those controls on this platform. The list of supported Control IDs can be found in TABLE 30: NUMERICALLY SORTED CONTROL ID ENUMERATIONS and TABLE 31: TEMPERATURE (TSDD) USAGE ENUMERATION

3.1.4.1 Get Available Controls (GACI)

The GACI object is a control detail object which is implemented by the BIOS that allows for retrieving the entire list of Control IDs supported by the BIOS (see ENUMERATIONS). Any Control ID that is present in the list is assumed to be a Control ID that is handled by the SW SMI REAL-TIME COMMUNICATIONS

INTERFACE read and write routines (section 3.3). The XTU Software will then attempt to use the ACPI RUN-TIME CONTROL OBJECTS (section 3.1.5) in order to read whether each Control ID is supported via these interfaces. The GACI object is responsible to communicate the static information for all Controls which are able to be manipulated on the platform.

Syntax for Signature

```
Method(GACI, 0, NotSerialized, 0, PkgObj)
```

Description

The purpose of this method is to retrieve basic data about controls that are supported by the BIOS.

Arguments

No input is required for this method.

Result

A package object is returned with the following definition:

```
Name (RETV, Package()
{     // Field Name                  // Field Type
      ErrorCode                     // DWORD
      DataBuffer                    // ControlIdData[]
})
```

The resultant buffer is defined as an array of packed ControlIdData C-structs.

```
struct ControlIdData
{
      DWORD ControlId
      WORD  NumberOfValues
      BYTE  Precision
      BYTE  Flags
      DWORD DefaultDataValue
      DWORD MinDataValue
      DWORD MaxDataValue
      DWORD MinDisplayValue
      DWORD MaxDisplayValue
}
```

Result Parameter Definitions

Field Name	Definition
ErrorCode	Defined as: Success == 0 Unexpected Error == 0xFFFFFFFF Any value that is returned which is not equal to 0 is considered a failure. In the failure case, the buffer is defined as indeterminate and the caller should not use that data.
DataBuffer	The buffer returned as part of the GACI call is an array of ControlIdData C-structs. It is valid to return an empty buffer. This would imply that only monitoring features are supported by the platform.

Table 5: GACI Return Value Definition

Field Name	Definition
ControlId	This field describes a Control ID that is supported by the BIOS via the SW SMI Real-Time Communications Interface.
NumberOfValues	This field is used for two purposes. First if the control requires a value set of discrete numbers as opposed to a set of continuous numbers then this field should be set to FFFFFFFh. This tells the XTU software to use the Discrete Supported Values (GDSV) method in order to retrieve the value set for this Control ID. If the Control ID can be described by a continuous set of values then this parameter describes the number of supported values that are contained in that data set. This allows the caller to determine the step size for both the Data Values and the Display Values in order to generate a complete data set as well as a complete set of options to display to the end-user.
Precision	This field is used to allow the BIOS to represent non-whole numbers as fixed-point values. The precision specified will be applied to all Display Values in the data set of the associated Control ID. The precision field will be used for both continuous and discrete value sets. See the following examples: DisplayValue: 125 Precision: 2 XTU UI: 1.25 DisplayValue: 40 Precision: 0 XTU UI: 40 DisplayValue: 400 Precision: 1 XTU UI: 40.0
Flags	Flag Bit Definitions: Bit[0] 1 – Real Time ACPI Interface Support 0 – No Real Time ACPI Interface Support If bit 0 is a 1 then the Run-Time Control Objects are implemented for this Control ID. Bit[1] 1 – Supports dynamic control mode 0 – Does not support dynamic control mode Special rules apply to Dynamic Control Mode Support. Bit[2:7] Reserved – Should be 00h
DefaultDataValue	The value of the data associated with the default setting for this Control ID. This data value must be contained within the value set described by the Min/Max Data Values or by the Discrete Supported Values (GDSV) data values.

Field Name	Definition
MinDataValue	The value of the data associated with the **MinDisplayValue**. This data will be sent to both the SW SMI REAL-TIME COMMUNICATIONS INTERFACE and the RUN-TIME CONTROL OBJECTS if they are supported when attempting to apply the minimum display value. This value is not used if the DISCRETE SUPPORTED VALUES (GDSV) method is implemented for this Control ID.
MaxDataValue	The value of the data associated with the **MaxDisplayValue**. This data will be sent to both the SW SMI REAL-TIME COMMUNICATIONS INTERFACE and the RUN-TIME CONTROL OBJECTS if they are supported when attempting to apply the maximum display value. This value is not used if the DISCRETE SUPPORTED VALUES (GDSV) method is implemented for this Control ID.
MinDisplayValue	The minimum value that is to be used for display purposes by the XTU user interface. This field is handled as a 2's complement to represent a negative value where necessary. This value is ignored if the DISCRETE SUPPORTED VALUES (GDSV) method is implemented for this Control ID. NOTE: This value can also be used for the non-standard data type definitions outlined below (see ENUMERATIONS): Enable/Disable Control IDs – In this case the MinDisplayValue should be 0. This represents the Disable state. XMP Profiles – In this case the MinDisplayValue should be 0. A DisplayValue of 0 represents Default Profile. A DisplayValue of 1 represents the Custom Profile. A DisplayValue of 2 represents Profile 1. A Display Value of 3 represents Profile 2. All other values are unsupported.
MaxDisplayValue	The maximum value that is to be used for display purposes by the XTU user interface. This field is handled as a 2's complement to represent a negative value where necessary. This value is ignored if the DISCRETE SUPPORTED VALUES (GDSV) method is implemented for this Control ID. NOTE: This value can also be used for the non-standard data type definitions outlined below (see ENUMERATIONS): Enable/Disable Control IDs – In this case the MinDisplayValue should be 0. This represents the Disable state. XMP Profiles – In this case the MinDisplayValue should be 0. A DisplayValue of 0 represents Default Profile. A DisplayValue of 1 represents the Custom Profile. A DisplayValue of 2

Field Name	Definition
	represents Profile 1. A Display Value of 3 represents Profile 2. All other values are unsupported.

Table 6: ControlIdData Structure Definition

3.1.4.1.1 Dynamic Control Mode Support

Dynamic control mode, when enabled, exposes a special supported value that is displayed as "Auto" in the XTU user interface. An example of a control that could support dynamic control mode would be CPU Core Voltage, where BIOSes will usually the display dynamic control mode setting as "Default" or "Automatic" via the BIOS set-up screens; while set to this value, the voltage regulators would automatically adjust the voltage in real-time based on load, thermal margins, and capabilities.

To indicate a control supports dynamic control mode, the corresponding flag must be set in the ControlIdData flags field for the control in the GET AVAILABLE CONTROLS (GACI) ACPI method. A control with this flag set to enabled will automatically assume that a supported value exists with a data value of 0xFFFFFFFE that will be displayed as "Auto" in the XTU user interface. **A BIOS must *not* include a setting with a data value of 0xFFFFFFFE when the dynamic control mode support flag is enabled;** that setting will be replaced with the "Auto" setting by the XTU application framework.

When a control has the dynamic control mode support flag enabled, all real-time devices, including those defined via ACPI, will be ignored. To summarize: controls supporting dynamic control mode **do not support real-time / run-time changes.**

The following table shows a listing of the control IDs that do not support dynamic control mode; **the dynamic control mode support flag will be ignored on any control that is listed on the following table.**

Controls that Do Not Support Dynamic Control Mode

Control ID	Definition
01h	Host Clock Frequency
1Dh	1-Active Core Ratio Limit
1Eh	2-Active Core Ratio Limit
1Fh	3-Active Core Ratio Limit
20h	4-Active Core Ratio Limit
2Ah	5-Active Core Ratio Limit
2Bh	6-Active Core Ratio Limit
60h	7-Active Core Ratio Limit
61h	8-Active Core Ratio Limit
00h	Maximum Non-Turbo Ratio for Processor Core
3Bh	Processor Graphics Slice Turbo Ratio Limit
65h	Processor Graphics Unslice Turbo Ratio Limit

49h	Memory Clock Multiplier
13h	DDR Multiplier
45h	Reference Clock Ratio
3Fh	Runtime Turbo Override
40h	XMP Profile Selection
29h	Enhanced Intel® SpeedStep® Technology Enable/Disable
47h	Runtime Turbo Override Enable
37h	Processor Graphics Core Total Design Power Enable
34h	IA Core Total Design Power Enable
1Ah	Turbo Boost Technology Enable
31h	Short Window Package Total Design Power Enable
48h	Configurable TDP Enable
50h	Overclocking Enable
4Ah	Filter PLL Frequency
4Bh	Dynamic SVID Control
4Ch	Ring Ratio
4Eh	Ring Voltage Mode
52h	Graphics Slice Voltage Mode
62h	Graphics Unslice Voltage Mode
54h	Package Current Limit
56h	FIVR Faults
57h	FIVR Efficiency Management
58h	IA Core Voltage Mode
5Ah	PEG/DMI Ratio
5Eh	ED RAM PLL Ratio
5Fh	Thermal Throttle Offset

Table 7: Controls that Do Not Support Dynamic Control Mode

3.1.4.2 Discrete Supported Values (GDSV)

The GDSV object is a control detail object which retrieves a specified Control ID's (see ENUMERATIONS) discrete set of BIOS setting values, display values, and an associated precision for the entire list. This mechanism is only necessary if either the display values or the setting values are non-continuous. This method also returns the precision of the display values.

Syntax for Signature

```
Method(GDSV, 1, NotSerialized, 0, PkgObj, IntObj)
```

Description
The purpose of this method is to retrieve the complete set of discrete values supported for the requested Control ID on this platform.

Arguments
The single input to the GDSV method is the Control ID to be queried.

Parameter Definitions

Field Name	Definition
ControlID (Arg0)	This is a value which represents a specified control (see ENUMERATIONS).

Table 8: GDSV Argument Definition

Result
A package object is returned with the following definition:

```
Name (RETV, Package()
{     // Field Name                  // Field Type
      ErrorCode                     // DWORD
      DataBuffer                    // DiscreteValueData[]
})
```

The resultant buffer is defined as an array of packed DiscreteValueData C-structs.

```
struct DiscreteValueData
{
      DWORD DataValue
      DWORD DisplayValue
}
```

Result Parameter Definitions

Field Name	Definition
ErrorCode	Defined as: Success == 0 Only Continuous Values Supported == 1 Unexpected Error == 0xFFFFFFFF Any value that is returned which is not equal to 0 is considered a failure. A value of 1 describes a Control ID whose data is only defined in the GET AVAILABLE CONTROLS (GACI). In any error condition the caller should not use the DataBuffer as its values are indeterminate.
DataBuffer	The buffer returned as part of the GDSV call is an array of DiscreteValueData C-structs. This array of structures should explicitly define all supported values for the requested Control ID. If both the SW SMI REAL-TIME COMMUNICATIONS INTERFACE and the RUN-TIME

Field Name	Definition
	CONTROL OBJECTS are supported, then the array of supported values will be shared between them. It is not valid to return an empty buffer.

Table 9: GACI Return Value Definition

Field Name	Definition
DataValue	This value will be sent as the input to both the SW SMI REAL-TIME COMMUNICATIONS INTERFACE and the RUN-TIME CONTROL OBJECTS for the associated **DisplayValue**.
DisplayValue	The value for the graphical user interface display which will be presented to the end-user. Any precision that is applied to the **DisplayValue** is described in the GET AVAILABLE CONTROLS (GACI) method with the associated Control ID. This field is handled as a 2's complement to represent a negative value where necessary.

Table 10: DiscreteValueData Structure Definition

3.1.4.3 Get SMI Command Value (GSCV)

The GSCV object is a control detail object which evaluates to the SMI command that should be sent to the appropriate SW SMI port for the platform. This is a custom value for each BIOS that designates which value should be placed in the AL register prior to performing the SW SMI described in the SW SMI REAL-TIME COMMUNICATIONS INTERFACE section of the document.

Syntax for Signature

```
Name (GSCV, 0xXX)
```

3.1.4.1 Get XMP Display Values (GXDV)

The GXDV object is a control detail object which retrieves the requested XMP profile's settings and their associated display values. This mechanism is only necessary if the platform supports XMP. It is an optional method for implementation. However it is required to be implemented for XTU to support XMP.

Syntax for Signature

```
Method(GXDV, 1, NotSerialized, 0, PkgObj, IntObj)
```

Description

The purpose of this function is to query the BIOS about the memory frequency, timings, and voltages associated with a specific XMP profile. All Memory settings that are supported by the platform must be returned as part of the BIOS Settings Data Structure returned from the SMI call. This includes every supported Control ID from the memory section of TABLE 30: NUMERICALLY SORTED CONTROL ID ENUMERATIONS as well as the Memory Voltage and optionally the System Agent Voltage from the voltage section of the enumeration. All of this data must be returned by this method.

Arguments

The single input to the GXDV method is the XMP Profile to be queried, Profile 1 or Profile 2.

Parameter Definitions

Field Name	Definition
ProfileNumber (Arg0)	This is a value which represents either Profile 1 or Profile 2. 1 – Retrieve values for Profile 1 2 – Retrieve values for Profile 2 All other inputs – Invalid and should return an error.

Table 11: GXDV Argument Definition

Result
A package object is returned with the following definition:

```
Name (RETV, Package()
{     // Field Name                  // Field Type
      ErrorCode                     // DWORD
      DataBuffer                    // XmpDisplayValue[]
})
```

The resultant buffer is defined as an array of packed XmpDisplayValue C-structs.

```
struct XmpDisplayValue
{
      WORD  ControlID
      BYTE  Reserved
      BYTE  Precision
      DWORD DisplayValue
}
```

Result Parameter Definitions

Field Name	Definition
ErrorCode	Defined as: Success == 0 Invalid Input Argument == 1 XMP Not Supported == 2 Unexpected Error == 0xFFFFFFFF Any value that is returned which is not equal to 0 is considered a failure. A value of 1 describes an invalid input. This is generally because a request for Profiles other than 1 & 2. In any error condition the caller should not use the DataBuffer as its values are indeterminate.

Field Name	Definition
DataBuffer	The buffer returned as part of the GXDV call is an array of XmpDisplayValue C-structs. This array of structures should explicitly define all Control IDs and their associated Display Values that will be altered if the requested XMP Profile is applied to the system. It is not valid to return an empty buffer.

Table 12: GXDV Return Value Definition

Field Name	Definition
ControlID	This field describes a Control ID that is manipulated if the currently queried XMP Profile is selected to be applied to the system.
Reserved	This field must be set to 00h.
Precision	This field is used to allow the BIOS to represent non-whole numbers as fixed-point values. The precision specified will be applied to the value in the **DisplayValue** field of this structure. See the following examples: DisplayValue: 125 Precision: 2 XTU UI: 1.25 DisplayValue: 40 Precision: 0 XTU UI: 40 DisplayValue: 400 Precision: 1 XTU UI: 40.0
DisplayValue	The value for the graphical user interface display which will be presented to the end-user.

Table 13: XmpDisplayValue Structure Definition

3.1.5 Run-Time Control Objects

The ACPI Control objects provide access to various voltage, clock, and other platform specific controls that are implemented by the BIOS on a specific platform. These objects can be accessed from the OS level to provide applications with access to manipulating certain types of hardware on the platform. This control is accomplished by defining and implementing ACPI device objects in the platform BIOS according to this specification and accessing them through the XTU software.

3.1.5.1 Control Device Read (CDRD)

The CDRD object is a control method which is implemented by the BIOS that allows for reading the current value of an object which is controllable in real-time. This object is only required to be implemented when supporting real-time control for platform specific hardware. Handlers to support Control IDs for Intel silicon based features are not required.

An example implementation can be found in Section 3.1.7.

Syntax for Signature

```
Method(CDRD, 1, Serialized, 0, PkgObj, IntObj)
```

Description

The purpose of this method is to be able to read the current value of the hardware via a BIOS implemented custom interface. This method will always provide the data necessary to determine the current value of the actual platform hardware.

Arguments

The CDRD control method has one input argument. The sole input is the Control ID that should be read.

Parameter Definitions

Field Name	Definition
ControlID (Arg0)	This is a value which represents a specified control (see ENUMERATIONS).

Table 14: CDRD Argument Definition

Result

A package object is returned with the following definition:

```
Name (RETV, Package()
{      // Field Name            // Field Type
       ErrorCode,               // DWORD
       DataValue                // DWORD
})
```

Result Parameter Definitions

Field Name	Definition
ErrorCode	Defined as: Success == 0 Unexpected Error == 0xFFFFFFFF Any value that is returned which is not equal to 0 is considered a failure to read the device. In this case, the value of the **DataValue** field is defined as indeterminate and the caller should not use that data.
DataValue	The current value of the hardware as reported by the BIOS. The meaning of these values is defined by either the GET AVAILABLE CONTROLS (GACI) or the DISCRETE SUPPORTED VALUES (GDSV) method.

Table 15: CDRD Return Value Definition

3.1.5.2 Control Device Write (CDWR)

The CDWR object is a control method which is implemented by the BIOS that allows for writing to an object which is controllable in real-time. This object is only required to be implemented when supporting real-time control for platform specific hardware. Handlers to support Control IDs for Intel silicon based

features are not required. The detailed list is in the following table. Please refer to Table 30 for platform compatibility issue. Not every control is available on each platform.

Control IDs	Definition
01h	Reference Clock Frequency
02h	CPU Voltage Override
1Dh	1-Active Core Ratio Limit
1Eh	2-Active Core Ratio Limit
1Fh	3-Active Core Ratio Limit
20h	4-Active Core Ratio Limit
2Ah	5-Active Core Ratio Limit
2Bh	6-Active Core Ratio Limit
60h	7-Active Core Ratio Limit
61h	8-Active Core Ratio Limit
22h	CPU Voltage Offset
2Eh	Additional Turbo Mode CPU Voltage
2Fh	Short Window Package Total Design Power Limit
30h	Extended Window Package Total Design Power Limit
31h	Short Window Package Total Design Power Enable
39h	IA Core Current Maximum
3Ah	Processor Graphics Slice Current Maximum
3Bh	Processor Graphics Slice Ratio Limit
3Ch	Additional Process Graphics Voltage
51h	Processor Graphics Slice Voltage Override
52h	Processor Graphics Slice Voltage Mode
53h	Processor Graphics Slice Voltage Offset
62h	Processor Graphics Unslice Voltage mode
63h	Processor Graphics Unslice Voltage Override
64h	Processor Graphics Unslice Voltage Offset
65h	Processor Graphics Unslice Ratio Limit
4Ch	Ring Ratio
4Dh	Ring Voltage Override
4Eh	Ring Voltage Mode

4Fh	Ring Voltage Offset
43h	Short Window Time
42h	Extended Window Time
3Fh	Runtime Turbo Override
54h	Package Current Limit
56h	FIVR Fault
57h	FIVR Efficiency management
58h	IA Core Voltage Mode

Table 16: Control Ids with Runtime Support Built-In

An example can be found in Section 3.1.7.

Syntax for Signature

```
Method(CDWR, 2, Serialized, 0, IntObj, {IntObj, IntObj})
```

Description

The purpose of this method is to be able to write the requested value to the hardware via a BIOS implemented custom interface. This method is responsible to write the requested value to hardware and return a success or fail status to the caller.

Arguments

The CDWR control method has two input arguments. Both arguments are DWORD values. The first argument is the Control ID. The second argument is the value to be written to the hardware.

Parameter Definitions

Field Name	Definition
ControlID (Arg0)	This is a value which represents a specified control (see ENUMERATIONS).
DataValue (Arg1)	The value that is being requested to be written to hardware. It will be a value the XTU software has retrieved from either the GET AVAILABLE CONTROLS (GACI) or the DISCRETE SUPPORTED VALUES (GDSV) methods.

Table 17: CDWR Argument Definition

Result

```
Name (RETV, ErrorCode)
```

Result Parameter Definitions

Field Name	Definition
ErrorCode	Defined as: Success == 0

Field Name	Definition
	Non-real Time Control ID requested = 1 Unexpected Error == 0xFFFFFFFF Any value that is returned which is not equal to 0 is considered a failure to write to the device.

3.1.6 Monitor-Only Objects

The Monitor-Only objects provide access to various temperature, voltage, and fan data implemented by BIOS on a particular platform through an application level mechanism. This is accomplished by defining and implementing the methods described in this section within the platform BIOS and using software (i.e. Intel® Extreme Tuning Utility) to view the thermal data.

3.1.6.1 Temperature Sensor Data Dump (TSDD)

The TSDD method evaluates to a packaged list of information about available temperature sensors and the current absolute temperature values. This object is required to be implemented when using any Performance Tuning & Monitoring ACPI Devices.

Typical temperature values returned by this object would include processor diode temperature (if available and accessible). Other platform temperature sensors like voltage regulator, memory, or notebook skin may also be returned.

Syntax for Signature

```
Method(TSDD, 0, NotSerialized, 0, PkgObject)
```

Description

The purpose of this method is to be able to get the current state of all temperatures on the platform which have been provided by the platform.

Arguments

No input parameters.

Result

```
Name (RETV, Package()
{     //Field Name            //Field Type
      UsageId1,               // DWORD
      UniqueId1,              // DWORD
      CurrentValue1           // DWORD
      Reserved1,              // DWORD
      …
      …
      UsageIdN,               // DWORD
      UniqueIdN,              // DWORD
      CurrentValueN,          // DWORD
      ReservedN               // DWORD
})
```

NOTE: If no temperature sensors are present on the system, then a null package must be returned for the TSDD object.

Result Parameter Definitions

Field Name	Definition
UsageId	Indicates the type of device the temperature value is reported for. The value must be one of the values from TABLE 31: TEMPERATURE (TSDD) USAGE ENUMERATION.
UniqueId	The UniqueId value reported by BIOS in the TSDD package must uniquely identify a device within the Performance Tuning & Monitoring ACPI Device scope (this includes VSDD and FSDD devices as well).
CurrentValue	The units of the current absolute temperature value returned must be 10ths of a Kelvin. For example, if the temperature is 30 degrees Celsius then the value returned must be (2732 + 300) = 3032.
Reserved	The value of the reserved field is 0000h.

Table 18: TSDD Package Parameter Definitions

3.1.6.2 Voltage Sensor Data Dump (VSDD)

The VSDD method evaluates to a packaged list of information about available voltage sensors and the current voltage values. This object is required when using any Performance Tuning & Monitoring ACPI Devices.

Typical voltage values returned by this object would include CPU core, Uncore, Memory, and/or PCH.

Syntax for Signature

```
Method(VSDD, 0, NotSerialized, 0, PkgObject)
```

Description

The purpose of this method is to be able to get the current state of all voltages on the platform which have been provided by the platform.

Arguments

No input parameters.

Result

```
Name (VLTV, Package()
{     //Field Name              //Field Type
      UsageId1,                 // DWORD
      UniqueId1,                // DWORD
      CurrentValue1             // DWORD
      Reserved1,                // DWORD
      …
      …
      UsageIdN,                 // DWORD
      UniqueIdN,                // DWORD
      CurrentValueN,            // DWORD
      ReservedN                 // DWORD
})
```

NOTE: If no voltages are present on the system, then a null package must be returned for the VSDD object.

Result Parameter Definitions

Field Name	Definition
UsageId	Indicates the type of device the voltage value is reported for. UsageId for VSDD package must be one of the values from TABLE 32: VOLTAGE (VSDD) USAGE ENUMERATION.
UniqueId	The UniqueId value reported by BIOS in the VSDD package must uniquely identify a device within the Performance Tuning & Monitoring ACPI Device scope (this includes TSDD and FSDD devices as well).
CurrentValue	The unit of the current voltage returned must be millivolts (mV). E.g., if the Voltage is 1.1 V, then the value returned must be 1100.
Reserved	The value of the reserved field is 0000h.

Table 19: VSDD Package Parameter Definitions

3.1.6.3 Fan Sensor Data Dump (FSDD)

The FSDD method evaluates to a packaged list of information about available fan sensors and the current fan speed values. This object is required when using any Performance Tuning & Monitoring ACPI Devices.

Syntax for Signature

```
Method(FSDD, 0, NotSerialized, 0, PkgObject)
```

Description

The purpose of this method is to be able to get the current speed of all fans on the platform which have been provided by the platform.

Arguments

No input parameters.

Result

```
Name (RPMV, Package()
{     //Field Name              //Field Type
      UsageId1,                 // DWORD
      UniqueId1,                // DWORD
      CurrentValue1,            // DWORD
      Reserved1,                // DWORD
      …
      …
      UsageIdN,                 // DWORD
      UniqueIdN,                // DWORD
      CurrentValueN,            // DWORD
      ReservedN                 // DWORD
})
```

NOTE: If no fan sensors are present on the system, then a null package must be returned.

Result Parameter Definitions

Field Name	Definition
UsageId	Indicates the type of device the fan speed value is reported for. UsageId for FSDD package must be one of the values from TABLE 33: FAN (FSDD) USAGE ENUMERATION.
UniqueId	The UniqueId value reported by BIOS in the FSDD package must uniquely identify a device tithing the Performance Tuning & Monitoring ACPI Device scope (this includes TSDD and VSDD devices as well).
CurrentValue	The unit of the current fan speed returned must be rotations per minute (RPM). E.g., if the speed is 2500 RPM, then the value returned must be 2500.
Reserved	The value of the reserved field is 0000h.

Table 20: FSDD Package Parameter Definitions

3.1.6.4 Sensor Data Sampling Period (SDSP)

This optional object evaluates to an integer to specify the sampling period to evaluate TSDD, VSDD and FSDD methods that would guarantee fresh data for temperature, voltage and fan speed values. The unit of sampling is in 10ths of seconds.

For example, in a platform that has one temperature sensor, one voltage sensor and one fan speed sensor, if hardware implementation takes 100 ms (0.1 s), 200 ms (0.2 s) and 500 ms (0.5 s) to fetch temperature, voltage and fan speed values, then the SDSP must return 5.

When this method is present, the OS/application level software should honor the value returned by this object. The OS/Application level software can evaluate the TSDD, VSDD and FSDD objects at a sampling rate of the period specified by this object or above.

Syntax for Signature

```
Method(SDSP, 0, NotSerialized, 0, IntObject)
```

Description

The purpose of this method is to get the recommended sampling period for the platform temperatures, voltages, and fans.

Arguments

No input parameters.

Result

```
Name(RETV, SamplingPeriod)
```

Result Parameter Definitions

Field Name	Definition
SamplingPeriod	Indicates the minimum sampling period that the application can use and expect to receive updated information from the platform for the TSDD, FSDD, and VSDD methods.

Table 21: SDSP Result Parameter Definitions

3.1.7 Example Implementation

First is the definition of BIOS POST time C-struct definitions and initialization.

```
//
// GACI structure definition
//
typedef struct ControlIdData
{
   UINT32   ControlId;
   UINT16   NumberOfValues;
   UINT8    Precision;
   UINT8    Flags;
   UINT32   DefaultDataValue;
   UINT32   MinDataValue;
   UINT32   MaxDataValue;
   UINT32   MinDisplayValue;
   UINT32   MaxDisplayValue;
} CONTROLID_DATA;

#DEFINE SUPPORTED_CONTROLID_COUNT 6 // Count of 6 is an example
typedef struct CtlBufer
{
      CONTROLID_DATA CtrlID[SUPPORTED_CONTROLID_COUNT];
} CONTROLID_BUFF;


STATUS CreateGaciBuffer (VOID)
{

.
.
.

CONTROLID_BUFF    *CtlBuf;

AllocateMemory(EfiACPIMemoryNVS, sizeof(CONTROLID_BUFF), &CtlBuf);

CtlBuf->CtrlID[0].ControlId         = 0x00;
CtlBuf->CtrlID[0].NumberOfValues    = MaxNonTurboRatio - MaxEffRatio+1;
CtlBuf->CtrlID[0].Precision         = 0x00;
CtlBuf->CtrlID[0].Flags             = 0x00;
CtlBuf->CtrlID[0].DefaultDataValue  = FlexRatioOverrideDefault;
CtlBuf->CtrlID[0].MinDataValue      = MaxEfficiencyRatio;
CtlBuf->CtrlID[0].MaxDataValue      = MaxNonTurboRatio;
CtlBuf->CtrlID[0].MinDisplayValue   = MaxEfficiencyRatio;
CtlBuf->CtrlID[0].MaxDisplayValue   = MaxNonTurboRatio;
```

```
CtlBuf->CtrlID[1].ControlId          = BIOS_DEVICE_HOST_CLK_FREQ;
CtlBuf->CtrlID[1].NumberOfValues     = BclkMaxValue - BclkMinValue + 1;
CtlBuf->CtrlID[1].Precision          = 0x02;
CtlBuf->CtrlID[1].Flags              = 0x00;
CtlBuf->CtrlID[1].DefaultDataValue   = 10000;
CtlBuf->CtrlID[1].MinDataValue       = BclkMinValue;
CtlBuf->CtrlID[1].MaxDataValue       = BclkMaxValue;
CtlBuf->CtrlID[1].MinDisplayValue    = BclkMinValue;
CtlBuf->CtrlID[1].MaxDisplayValue    = BclkMaxValue;

CtlBuf->CtrlID[3].ControlId          = BIOS_DEVICE_tCL;
CtlBuf->CtrlID[3].NumberOfValues     = tCL_MAX - tCL_MIN +1;
CtlBuf->CtrlID[3].Precision          = 0x00;
CtlBuf->CtrlID[3].Flags              = MIN_SETTING_LOW_PERFORMANCE;
CtlBuf->CtrlID[3].DefaultDataValue   = tCLDefault;
CtlBuf->CtrlID[3].MinDataValue       = tCL_MIN;
CtlBuf->CtrlID[3].MaxDataValue       = tCL_MAX;
CtlBuf->CtrlID[3].MinDisplayValue    = tCL_MIN;
CtlBuf->CtrlID[3].MaxDisplayValue    = tCL_MAX;

CtlBuf->CtrlID[4].ControlId          = BIOS_DEVICE_tRCD;
CtlBuf->CtrlID[4].NumberOfValues     = tRCD_MAX - tRCD_MIN + 1;
CtlBuf->CtrlID[4].Precision          = 0x0;
CtlBuf->CtrlID[4].Flags              = MIN_SETTING_LOW_PERFORMANCE;
CtlBuf->CtrlID[4].DefaultDataValue   = tRCDDefault;
CtlBuf->CtrlID[4].MinDataValue       = tRCD_MIN;
CtlBuf->CtrlID[4].MaxDataValue       = tRCD_MAX;
CtlBuf->CtrlID[4].MinDisplayValue    = tRCD_MIN;
CtlBuf->CtrlID[4].MaxDisplayValue    = tRCD_MAX;


CtlBuf->CtrlID[5].ControlId          = BIOS_DEVICE_tRP;
CtlBuf->CtrlID[5].NumberOfValues     = tRP_MAX - tRP_MIN + 1;
CtlBuf->CtrlID[5].Precision          = 0x00;
CtlBuf->CtrlID[5].Flags              = MIN_SETTING_LOW_PERFORMANCE;
CtlBuf->CtrlID[5].DefaultDataValue   = tRPDefault;
CtlBuf->CtrlID[5].MinDataValue       = tRP_MIN;
CtlBuf->CtrlID[5].MaxDataValue       = tRP_MAX;
CtlBuf->CtrlID[5].MinDisplayValue    = tRP_MIN;
CtlBuf->CtrlID[5].MaxDisplayValue    = tRP_MAX;

.
.
.

}
```

The example below illustrates a sample implementation of the Performance Tuning & Monitoring ACPI device in ASL.

```
//
// Define the XTU Device as a dynamically loadable SSDT or within the
// DSDT under the \_SB scope
```

```
//

Scope (\_SB)
{
      // First declare external variables for items that need to be
      // fixed up during POST
      // The XTUB structure should point at the CtlBuf which was
      // allocated and populated during POST (see previous C-struct
      // example).
      External(XTUB)
      OperationRegion (XNVS, SystemMemory, XTUB, 0x2000)
      Field (XNVS, ByteAcc, NoLock, Preserve)
      {
            XBUF, 0x16c0      // GACI Size specific to implementation
      }

      // Note: When declaring the device, any name unique to the
      // platform implementation can be used (i.e. PTMD as below)
      Device(PTMD)
      {
            Name(_HID, EISAID("INT3394"))
            Name(_CID, EISAID("PNP0C02"))

            Name(IVER, 0x00010000)
            Name(GSCV, 0x10000)

            Method(GACI, 0x0, NotSerialized, 0, PkgObj)
            {
                  Name(RPKG, Package(0x2){})    // Return package
                  Store(0x00,Index(RPKG, 0))    // ErrorCode
                  Store(XBUF, Index(RPKG, 1))   // buffer

                  Return(RPKG)
            }

            Method(GDSV, 0x1, Serialized)
            {
                  // The next line represents checking for specifically
                  // supported ControlIDs.  Typically this would be a
                  // Case or If/ElseIf statement if multiple ControlIDs
                  // were supported.  The default condition should be
                  // an error code where the assumption is that
                  // Discrete values are not supported (or necessary)
                  // for the requested ControlID.
                  If(LEqual(Arg0, 0x07))
                  {
                        Return(Package(0x2)
                        {
                              Zero,             //Error Code
                              Buffer()
                              {
                              0x07,Zero,Zero,Zero,//Data Value 1
```

```
                        0x07,Zero,Zero,Zero,//Display Value 1

                        0x09,Zero,Zero,Zero,//Data Value 2
                        0x09,Zero,Zero,Zero //Display Value 2

                        0x0e,Zero,Zero,Zero,//Data Value 3
                        0x0e,Zero,Zero,Zero //Display Value 3
                        }
                  })
            }

            Return(Package(0x1)
            {
                  0x01  //Error code for continuous settings
            })
      }

      // This method is the Control Device Read
      // Arguments by number:
      //    0 - Method Name
      //    1 - Number of CDRD input parameters
      //    2 - Mutex Requirements (See the ACPI Spec)
      //    3 - SyncLevel (See the ACPI Spec)
      //    4 - Return Type
      //    5 - List of Input Types (1 Integer)
      Method(CDRD, 1, Serialized, 0, PkgObj, IntObj)
      {
            Return(Package(0x2)
            {
                  Zero, //Error Code
                  Zero  //Current Value
            })
      }

      // This method is the Control Device Write
      // Arguments by number:
      //    0 - Method Name
      //    1 - Number of CDWR input parameters
      //    2 - Mutex Requirements (See the ACPI Spec)
      //    3 - SyncLevel (See the ACPI Spec)
      //    4 - Return Type
      //    5 - List of Input Types (2 Integers)
      Method(CDWR, 0x2, Serialized, 0, IntObj, {IntObj, IntObj})
      {
            Return(Zero)      //Error Code
      }

      Name(TMPV, Package()
      {
            //UsageId   //UniqueId     //Value   //Reserved
            0x01,       0x0002,           0,          0,
            0x03        0x0003,           0,          0,
```

```
      0x06        0x0004,           0,          0
})

Name(VLTV, Package()
{
      //UsageId   //UniqueId     //Value   //Reserved
      0x01,       0x0005,          0,          0
      0x04        0x0006,          0,          0
      0x06        0x0007,          0,          0
      0x10        0x0008,          0,          0
})

Name(RPMV, Package()
{
      //UsageId   //UniqueId     //Value   //Reserved
      0x01,       0x0009,          0,          0
      0x04        0x000A,          0,          0
      0x0C        0x000B,          0,          0
})

Method(TSDD)
{
      Name(TMPC, 0)  // Current Temperature Local Variable

      // Implement temperature determination code here
      // E.g. If embedded controller firmware implements a
      // command to fetch various temperature values,
      // implement code to issue the command. Populate the
      // TMPV package with the right temperature values
      …
      …
      // Update CurrentValue1 in TMPV package with
      // the current Temperature
      Store(TMPC, Index(TMPV, 2))
      …
      …
      // Update CurrentValue2 in TMPV package with
      // the current Temperature
      Store(TMPC, Index(TMPV, 6))
      Return(TMPV)
}

Method(VSDD)
{
      Name(VLTC, 0)  // Current volts Local Variable

      // Implement voltage determination code here
      // E.g. If embedded controller firmware implements a
      // command to fetch various voltage values, implement
      // code to issue the command. Populate the LVTV
      // package with the right voltage values
      …
```

```
                        // Update CurrentValue1 in VLTV package with
                        // the current Voltage
                        Store(VLTC, Index(VLTV, 2))
                        …
                        // Update CurrentValue2 in VLTV package with
                        // the current Voltage
                        Store(VLTC, Index(VLTV, 6))
                        Return(VLTV)
                  }

                  Method(FSDD)
                  {
                        Name(RPMC, 0)  // Current RPM Local Variable

                        // Implement Fan speed RPM determination code here
                        // E.g. If embedded controller firmware implements a
                        // command to fetch various RPM values, implement
                        // code to issue the command.  Populate the RPMV
                        // package with the current fan speed
                        …
                        // Update CurrentValue1 in RPMV package with
                        // the current Fan Speed
                        Store(RPMC, Index(RPMV, 2))
                        …
                        // Update CurrentValue2 in RPMV package with
                        // the current Fan Speed
                        Store(RPMC, Index(RPMV, 6))
                        Return(RPMV)
                  }

                  Method(SDSP)
                  {
                        // Fastest sampling period supported
                        // Expressed in tenths of a second
                        Return(10)
                  }
            } // End of PTMD Device
      }
```

3.2 Watchdog Timer

The only watchdog timer (WDT) implementation that is supported by this revision of XTU is the WDT that is integrated into the PCH. In order to support the PCH-based Watchdog Timer which is present on Cougar Point-based platforms and newer, XTU BIOS support for the timer requires integration of the chipset reference code. This documentation is provided separately from the XTU BIOS Interface Specification and is available from your technical BIOS support contact at Intel. Aside from the integration of the reference code, no XTU-specific BIOS support code is necessary.

3.3 SW SMI Real-Time Communications Interface

3.3.1 Overview

The main purpose of the SW SMI Real-Time Communications Interface is to read and write BIOS settings. This interface uses values that are obtained via data retrieved from the GET AVAILABLE CONTROLS (GACI) method described earlier in the document. These functions can be accessed in the Operating System via writes of the SW SMI Command Value to the SW SMI Port with the appropriate register settings which are described below.

3.3.2 BIOS Settings Structure

This structure defines the data that will be described by reads and writes to the BIOS SW SMI command defined by this specification. The BIOS is required to check the signature field and the length Field prior to writing any data to the buffer provided by the calling application. If either the signature or the length fields are not correct the BIOS must respond accordingly:

- If the signature field is correct, the current revision is supported, and the length field is sufficient, then fill in all the data, update the length field, and return successful.
- If the signature is correct but either the length is not sufficient to return all data or the revision is not supported, then fill in the correct length, major and minor revision fields and return the appropriate error or warning code.
- If the signature is not correct and it is not recognized then do not write any data to the supplied buffer and return an error.

Data Structure:

Offset	Name	Length	Value
00h	Signature	DWORD	’$BD2’
04h	Length	DWORD	Varies
08h	Major Revision	WORD	2
0Ah	Minor Revision	WORD	0
0Ch	BIOS Setting Count	DWORD	Varies
10h	BIOS Setting Entry Array	Varies	Varies

Table 22: BIOS Settings Data Structure

Offset	Name	Length	Value
00h	Control ID	DWORD	Varies
04h	Data Value	DWORD	Varies

Table 23: BIOS Setting Entry

3.3.3 Functions

3.3.3.1 Read BIOS Settings

This function reads the value for all BIOS settings that are present on the interface and places them into a memory location pointed to by the caller. As stated in the overview, in order to access this function, XTU will write the SW SMI Command Value to the SW SMI Port. Prior to this the registers must be setup as described in the command data section. The BIOS Settings Data Structure on a read must contain a list of all values supported by the platform.

Command Data:
Note: BIOS must be able to address up to 4GB of physical memory from SMM to support this function.

Use the data structure defined in Table 22: BIOS Settings Data Structure.

Register	Value	Definition
ECX	00h	Read BIOS Settings Command
EBX	Varies	32-bit Physical Memory Data Location of the location to be used for the returned BIOS Settings Data Structure (See Table 22)

Table 24: Read BIOS Settings Command, Register Setup

3.3.3.2 Write BIOS Settings

This function writes all BIOS settings that are present on the interface based on the data contained in a memory location pointed to by the caller. As stated in the overview, in order to access this function, XTU will write the SW SMI Command Value to the SW SMI Port. Prior to the SMI invocation the registers must be setup as described in the command data section. The BIOS Settings Data Structure on a write command will only contain a list of values changed since the previous write.

Command Data:
Note: BIOS must be able to address up to 4GB of physical memory from SMM to support this function.

Use the data structure defined in Table 22: BIOS Settings Data Structure.

Register	Value	Definition
ECX	01h	Write BIOS Settings Command
EBX	Varies	32-bit Physical Memory Data Location of the location to be used for the BIOS Settings Data Structure (See Table 22) to be written.

Table 25: Write BIOS Settings Command, Register Setup

3.3.4 Return Values

This table contains a list of possible error codes that can be returned from the BIOS in the EBX register to indicate the status of the last SMI call**.**

3.3.4.1 Error Codes

These codes define the return values that indicate a critical failure occurred during the SMI call. For all critical error conditions the high bit of the DWORD will be set.

NOTE: For all Error Codes considered critical errors the high bit of the DWORD returned must be set.

Value	Definition
0x00	Successful
0x8001	Invalid Signature supplied by caller
0x8002	Table length is too small, valid header data returned
0x8003	Table length is too small, no header data returned
0x8004	Unknown Command in ECX
0x8006	Invalid SMI revision
0xFFFF	Internal BIOS error - used for BIOS errors that cannot be generically classified. Use ECX to return a value that will aid in debugging/explaining this return value in more detail. Any data contained in ECX when this code is returned is a BIOS specific value and is not defined by this specification.

Table 26: BIOS Settings Command Error Codes

3.3.4.2 Warning Codes

These codes define the return values that indicate some issue occurred with the call but the data was able to be returned. Each warning may indicate that a subset of the full data set was returned.

Value	Definition
0x0002	Table length is too large (non-critical error). A complete data set of the supported table will be returned.
0x00FF	Internal BIOS warning - used for BIOS warnings that cannot be generically classified. Use ECX to return a value that will aid in debugging/explaining this return value in more detail. Any data contained in ECX when this code is returned is a BIOS specific value and is not defined by this specification.

Table 27: BIOS Settings Command Warning Codes

4 Control Implementation Guidance

This section provides control implementation guidance **specific to the Skylake generation of processors**.

4.1 Graphics Turbo Ratio Limit

The graphics slice (3Bh) and unsliced (65h) turbo ratio limit represents the ratio of the processor graphics multiplier with respect to BCLK. While the hardware value of this control is generally stored in 50MHz step-sized values (as on HSW CPUs), the display value reported to XTU must be reported in 100MHz steps so that it is relative to the BCLK value.

For example, for a graphics frequency value of 1150MHz, the data value read directly from hardware is 21 or 15h but the display value must be reported to XTU as 11.5x.

4.2 Memory Clock Multiplier

The memory clock multiplier (control 49h) represents the ratio of the memory controller PLL reference clock to BCLK. For example, for the (Panther Point default) frequency value of 133MHz, this multiplier value would be 133MHz / 100MHz = 1.33.

Note that, if supported by the platform, the memory clock multiplier control should be included with the Get XMP Display Values (GXDV) ACPI Device Object for each supported XMP memory profile. Without this value, XTU is unable to calculate the operating memory frequency for the corresponding XMP profile, since the DDR Multiplier (control 13h) is relative to the memory controller frequency.

The memory controller reference clock frequency can be read on SKL from OC Mailbox command 0x3.

4.3 FIVR Voltage Controls

For each FIVR domain supported by the processor, XTU supports three controls:

- **Voltage mode.** An enumeration value that indicates whether the corresponding domain's override voltage should be applied in either "Static" or "PCU Adaptive" mode.
- **Override voltage.** A voltage value that is applied to the corresponding domain. Whether this voltage is applied as a static or adaptive value is determined by the corresponding "Voltage mode" setting. For ideal operation, the values allowed by this control should be a subset of those supported by the voltage override field of the system OC mailbox.
- **Voltage offset.** A fixed offset value to be applied by the PCU to the final computed voltage value for the given domain. This value corresponds to the "voltage offset" value provided to the OC mailbox. For ideal operation, the values allowed by this control should be a subset of those supported by the voltage offset field of the system OC mailbox.

Note that XTU expects that the Max OC ratio value programmed to the OC mailbox for the IA domain will match the value of the 1-Active Core Ratio Limit (1Dh) control value. If these do not match, XTU will use the lower of the two values.

FIVR is supported on HSW and BDW platforms.

4.4 PEG/DMI Ratio

The PEG/DMI Ratio (control 5Ah) represents the ratio of the PEG and DMI bus frequency to BCLK. For example, given an operating PEG/DMI frequency of 100MHz and a BCLK frequency of 125MHz, this multiplier value would be 100MHz / 125MHz = 0.8. Thus, one or more of the following ratios must be supported: 1.0 / 0.8 / 0.6 / 0.4.

On SKL platform, a separate clock is used for PEG/DMI on both CPU and PCH. This clock frequency is fixed at 100MHz. Therefore, the PEG/DMI ratio is not relevant on SKL.

4.5 Filter PLL Frequency

The Filter PLL implemented on platforms can be configured in one of two ways. Either the frequency is left to run at its default value, which may be 1600MHz or 3200MHz, or it is forced to operate at 1600MHz. To reflect this design, this control should be implemented as a dynamic control using a numeric value. The control will thus support the reserved value for "Automatic" mode, which corresponds to the 1600/3200 default previously described.

4.6 OC Mailbox Enable/Disable Toggle Controls

The following controls are implemented as single bit positions within the OC Mailbox interface.

- Dynamic SVID Control
- FIVR Faults
- FIVR Efficiency Management

Each of these controls are represented in the XTU user interface with a value of "Enabled" or "Disabled" based on the display value (0h/1h) presented within the XTU ACPI interface. This value, however, is the inverse of the value used by the corresponding bit field within the OC Mailbox. The following table summarizes the relationship between XTU display value, ACPI value, and OC mailbox bit field value for all of these controls.

Table 28: OC Mailbox Enable/Disable Toggle Controls

XTU Display Value	ACPI Display Value	OC Mailbox Bit field Value
Enabled	01h	00h
Disabled	00h	01h

This is valid for the platform with FIVR.

Appendix A - Enumerations

The following tables represent all of the Control IDs supported by the XTU application. The first table has the Control IDs organized by subsystem for easy ability to find the appropriate devices. A separate table follows which lists all Control IDs numerically (TABLE 30: NUMERICALLY SORTED CONTROL ID ENUMERATIONS).

Table 29: Usage Sorted Control ID Enumerations

Subsystem	Control IDs	Definition	Type	Units
Processor	00h	Max Non-Turbo Processor Multiplier (also known as Flex Ratio)	Numeric	None
	02h	CPU Voltage Override	Numeric	Volts
	1Ah	Turbo Mode Enable	En/Dis	None
	1Dh	1-Active Core Ratio Limit	Numeric	None
	1Eh	2-Active Core Ratio Limit	Numeric	None
	1Fh	3-Active Core Ratio Limit	Numeric	None
	20h	4-Active Core Ratio Limit	Numeric	None
	2Ah	5-Active Core Ratio Limit	Numeric	None
	2Bh	6-Active Core Ratio Limit	Numeric	None
	60h	7-Active Core Ratio Limit	Numeric	None
	61h	8-Active Core Ratio Limit	Numeric	None
	22h	Dynamic CPU Voltage Offset	Numeric	mV
	29h	Enhanced Intel® SpeedStep Technology Enable	En/Dis	None
	2Eh	Additional Turbo Mode CPU Voltage	Numeric	mV
	2Fh	Short Window Package Total Design Power Limit (PL2)	Numeric	Watts
	30h	Extended Window Package Total Design Power Limit (PL1)	Numeric	Watts
	43h	Short Window Time (Tau for PL2)	Numeric	Seconds
	42h	Extended Window Time (Tau for PL1)	Numeric	Seconds
	31h	Short Window Package Total Design Power Enable	En/Dis	None
	32h	Package Total Design Power Lock Enable	En/Dis	None
	33h	IA Core Total Design Power Limit	Numeric	Watts
	34h	IA Core Total Design Power Enable	En/Dis	None
	35h	IA Core Total Design Power Lock Enable	En/Dis	None

	39h	IA Core Current Maximum	Numeric	Amps
	3Ch	Additional Turbo Mode Graphics Core Voltage	Numeric	mV
	3Fh	Runtime Turbo Override	Numeric	None
	41h	Internal PLL Overvoltage Enable	En/Dis	None
	46h	Vboot Voltage	Numeric	Volts
	47h	Runtime Turbo Override Enable	En/Dis	None
	50h	Overclocking Enable	En/Dis	None
	54h	Package Current Limit	Numeric	Amps
	58h	IA Core Voltage Mode *Enumeration Definition:* • 0 – Adaptive • 1 – Static	Enum	None
	66h	Process Core Current Limit Maximum	Numeric	Amps
Graphics	36h	Processor Graphics Core Total Design Power Limit	Numeric	Watts
	37h	Processor Graphics Core Total Design Power Enable	En/Dis	None
	38h	Processor Graphics Core Total Design Power Lock Enable	En/Dis	None
	3Ah	Processor Graphics Core Current Maximum	Numeric	Amps
	3Bh	Processor Graphics Slice Turbo Ratio Limit	Numeric	None
	51h	Processor Graphics Core Voltage Override	Numeric	Volts
	52h	Processor Graphics Core Voltage Mode *Enumeration Definition:* • 0 – Adaptive • 1 – Static	Enum	None
	53h	Processor Graphics Core Voltage Offset	Numeric	mV
	62h	Processor Graphics Unslice Voltage mode *Enumeration Definition:* • 0 – Adaptive • 1 – Static	Enum	None
	63h	Processor Graphics Unslice Voltage Override	Numeric	mV
	64h	Processor Graphics Unslice Voltage Offset	Numeric	mV
	65h	Processor Graphics Unslice Turbo Ratio Limit	Numeric	None
	68h	Processor Graphics Slice Current Limit Maximum	Numeric	Amps

	69h	Processor Graphics Unslice Current Limit Maximum	Numeric	Amps
System Agent	25h	System Agent Voltage Override	Numeric	Volts
	55h	System Agent Voltage Offset	Numeric	mV
	67h	System Agent Current Limit Maximum	Numeric	Amps
Ring/CLR	4Ch	Ring Ratio (HSW and later)	Numeric	None
	4Eh	Ring Voltage Mode *Enumeration Definition:* • 0 – Adaptive • 1 – Static	Enum	None
	4Dh	Ring Voltage Override	Numeric	Volts
	4Fh	Ring Voltage Offset	Numeric	mV
	6Ah	Ring Current Limit Maximum	Numeric	None
Clocking	01h	Reference Clock Frequency	Numeric	MHz
	45h	Reference Clock Ratio	Numeric	None
	4Ah	Filter PLL Frequency	Numeric	MHz
	5Ah	PEG/DMI Ratio	Numeric	None
Voltage	05h	Memory Voltage	Numeric	Volts
	44h	Secondary Memory VR Voltage	Numeric	Volts
	26h	PCH Voltage	Numeric	Volts
	3Dh	CPU PLL Voltage	Numeric	Volts
	3Eh	CPU IO Voltage	Numeric	Volts
	46h	Vboot Voltage	Numeric	Volts
	4Bh	Dynamic SVID Control	En/Dis	None
	57h	FIVR Efficiency Management Enable	En/Dis	None
	56h	FIVR Fault Enable	En/Dis	None
	59h	SVID Voltage Override	Numeric	V
	5Bh	I/O Analog Voltage Offset	Numeric	mV
	5Ch	I/O Digital Voltage Offset	Numeric	mV
Memory	13h	DDR Multiplier	Numeric	None
	49h	Memory Clock Multiplier	Numeric	None
	07h	CAS Latency (tCL)	Numeric	Clocks
	08h	Row Address to Column Address Delay (tRCD)	Numeric	Clocks

	09h	Row Precharge Time (tRP)	Numeric	Clocks
	0Ah	Row Active Time (tRAS)	Numeric	Clocks
	0Bh	Write Recovery Time (tWR)	Numeric	Clocks
	15h	Minimum Refresh Recovery Time (tRFC)	Numeric	Clocks
	16h	Row Active to Row Active delay (tRRD)	Numeric	Clocks
	17h	Internal Write to Read Command Delay (tWTR)	Numeric	Clocks
	18h	System Command Rate Mode	Numeric	None
	19h	Read to Precharge delay (tRTP)	Numeric	Clocks
	27h	Row Cycle Time (tRC)	Numeric	Clocks
	28h	Four Active Window Delay (tFAW)	Numeric	Clocks
	2Ch	Average Periodic Refresh Interval (tREFI)	Numeric	Clocks
	2Dh	Minimum CAS Write Latency Time (tCWL)	Numeric	Clocks
	40h	XMP Profile Selection *Enumeration Definition:* • 0 – Default SPD Profile • 1 – Custom Timing Profile • 2 – XMP Profile 1 • 3 – XMP Profile 2	Enum	Profile
	5Dh	All Banks Row Pre-Charge Delay Time (tRPab)	Numeric	Clocks
	5Eh	ED RAM PLL Ratio	Numeric	None

Table 30: Numerically Sorted Control ID Enumerations and Platform Compatibility

ID	Name	Description	Platform Compatibility	Comments
00h	Max Non-Turbo Processor Multiplier	This is the max IA core ratio, a.k.a. FLEX RATIO that the core can execute without turbo.		This control is used to instruct BIOS to program FLEX_RATIO (MSR 194h [15:8] plus its enable bit (Bit[16]), which when set at the right time during POST can adjust Max Non-Turbo Ratio (MNTR) which is reported via MSR CEh[15:8] below its fused value.
01h	Reference Clock Frequency	This is the reference clock, a.k.a BCLK frequency. The default is 100MHz.		
02h	CPU Voltage Override	This is the voltage on the CPU core voltage rail.		OC MB Command 0x10/0x11 [19:8], Domain ID = IA Core (0).

ID	Name	Description	Platform Compatibility	Comments
05h	Memory Voltage	Voltage directly supplied to DIMM		It is mapped to MRC supported value which defined how much voltage is given to a set of DIMMs.
07h	CAS Latency (tCL)			
08h	Row Address to Column Address Delay (tRCD)			
09h	Row Precharge Time (tRP)			
0Ah	Row Active Time (tRAS)			
0Bh	Write Recovery Time (tWR)			
0Dh	PCI Express Frequency	This is PCIe bus frequency that can be configured by BIOS.		
0Eh	PCI Frequency	This is PCI bus frequency that can be changed by BIOS.		
13h	DDR Multiplier	This is the number combining with the memory clock frequency that is used to generate a certain memory frequency.		24@133MHz and 32@100MHz?
15h	Minimum Refresh Recovery Time (tRFC)			
16h	Row Active to Row Active delay (tRRD)			
17h	Internal Write to Read Command Delay (tWTR)			
18h	System Command Rate Mode	Memory latency		
19h	Read to Precharge delay (tRTP)			

ID	Name	Description	Platform Compatibility	Comments
1Ah	Turbo Boost Technology Enable	To enable Intel® Turbo Boost Technology.		MSR 1A0h[38] 1 – disable 0 – enable (default)
1Dh	1-Active Core Ratio Limit	This is the max turbo ratio limit for 1 core active.		MSR 1ADh[7:0] – locked by OC lock bit (MSR 194h [20]) on everything except IVB-E. It is allowed to change when CEh[28] == 1(which is not used on HSW and later). OC MB Command 0x10/0x11 [7:0], domain 0; locked by OC lock bit (MSR 194h [20]) PCI 0xFF:0x0A:0:0xD8 [7:0] on Patsburg, locked by its own lock bit [63] PCI 0xFF:0x1E:0:0xD8 [7:0] on Wellsburg, locked by its own lock bit [63] MCHBAR 0x5990 [7:0], locked by its own lock bit [63] P-state Target (MSR 199h [14:8]), “locked” when SpeedStep is disabled (MSR 1A0h [16] == 0)
1Eh	2-Active Core Ratio Limit	This is the max turbo ratio limit for 2 core active		MSR 1ADh[15:8] – locked by OC lock bit (MSR 194h [20]) on everything except IVB-E Allowed to change when CEh[28] == 1.
1Fh	3-Active Core Ratio Limit			MSR 1ADh[23:16] – locked by OC lock bit (MSR 194h [20]) on everything except IVB-E Allowed to change when CEh[28] == 1.
20h	4-Active Core Ratio Limit			MSR 1ADh[31:24] – locked by OC lock bit (MSR 194h [20]) on everything except IVB-E Allowed to change when CEh[28] == 1.

ID	Name	Description	Platform Compatibility	Comments
22h	CPU Voltage Offset	The extra voltage that is allowed in either direction (+ or -) on the core voltage rail.		OC MB Command 0x10/0x11 [31:21] in format S11.0.10V +/- 1V
25h	System Agent Voltage Override	The constant voltage on the SA voltage rail when the override mode is selected	Pre-HSW	OC MB Command 0x10/0x11 [19:8] in format U12.2.10V
26h	PCH Voltage	Voltage to PCH power rail?		
27h	Row Cycle Time (tRC)			
28h	Four Active Window Delay (tFAW)			
29h	Enhanced Intel® Speedstep Technology Enable/Disable	Enable or disable EIST		MSR 1A0h Bit[16]: 1 - enable EIST, 0 - disable EIST
2Ah	5-Active Core Ratio Limit		SNB-E and later	MSR 1ADh[39:32] – locked by OC lock bit (MSR 194h [20]) on everything except IVB-E Allowed to change when CEh[28] == 1.
2Bh	6-Active Core Ratio Limit		SNB-E and later	MSR 1ADh[47:40] – locked by OC lock bit (MSR 194h [20]) on everything except IVB-E Allowed to change when CEh[28] == 1.
2Ch	Average Periodic Refresh Interval (tREFI)			
2Dh	Minimum CAS Write Latency Time (tCWL)			
2Eh	Max Turbo Mode CPU Voltage	Additional Turbo Voltage for CPU Core.		Sandy Bridge and Ivy Bridge only. This is no longer used since HSW which uses control 0x02 instead.

ID	Name	Description	Platform Compatibility	Comments
2Fh	Short Window Package Total Design Power Limit	This is the short term total design power limit (PL2). Typically this is 1.25 * the package TDP.		MSR 0x610 [46:32], PCI 0xFF:0xA:0:0xEC on Patsburg PCI 0xFF:0x1E:0:0xEC on Wellsburg On Server, the PCI space is used as the alternate to MSRs for power limits as well as the overall core ratio limit. MCHBAR 0x59A4 On Client, MMIO is used as an alternate to MSRs. The limit could be set in multiple places. The lowest limiter is the active value at runtime.
30h	Extended Window Package Total Design Power Limit	a.k.a long term power limit (PL1). Typically this is 1.0 * the package TDP		MSR 0x610 [14:0], PCI 0xFF:0xA:0:0xE8 on Patsburg, PCI 0xFF:0x1E:0:0xE8 on Wellsburg, MCHBAR 0x59A0
31h	Short Window Package Total Design Power Enable	To enable PL2	Excluding HSW-E	MSR 0x610 [47], PCI 0xFF:0xA:0:0xE8
32h	Package Total Design Power Lock Enable	When set, all power limits cannot be changed. This typically is set by BIOS to prevent runtime power limit manipulation.		MSR 0x610 [63], PCI 0xFF:0xA:0:0xE8
33h	IA Core Total Design Power Limit	This is the power limit for IA core power plane.		MSR 0x638 [12:0]
34h	IA Core Total Design Power Enable	This must be set in order to limit the power of the IA cores power plane		MSR 0x638 [15]
35h	IA Core Total Design Power Lock Enable	This must be set to prevent manipulation of IA core power plane power limit.		MSR 0x638 [31]
36h	Processor Graphics Core Total Design Power Limit	This is the power limit for the processor graphics power plane.	Pre-IVB only	MSR 0x640 [12:0]

ID	Name	Description	Platform Compatibility	Comments
37h	Processor Graphics Core Total Design Power Enable	This must be set in order to limit the power of the processor graphics power plane		
38h	Processor Graphics Core Total Design Power Lock Enable	This must be set to prevent manipulation of the processor graphics power plane power limit.		
39h	IA Core Current Maximum	Max current that CPU can draw from the platform.	SNB(-E) and IVB(-E) only	MSR 0x601 [12:0] This is only available on server parts. BIOS can program this register.
3Ah	Processor Graphics Core Current Maximum		Pre-IVB only	MSR 0x602 [12:0]
3Bh	Processor Graphics Slice Turbo Ratio Limit	The max turbo ration limit for GT slice		OC MB GT SLICE 0x10/11 [7:0], MCHBAR 0x5998
3Ch	Additional Turbo Mode Graphics Core Voltage	?	Pre-HSW only	
3Dh	CPU PLL Voltage	?		
3Eh	CPU IO Voltage	?		
3Fh	Runtime Turbo Override	?	SNB (-E) only	PCI [0xFF:0xA:0:0xD8]
40h	XMP Profile Selection	Select one of two possible XMP profiles from SPD		
41h	Internal PLL Overvoltage Enable	CPU voltage control. Filtering PLL voltage on SNB to enable/disable that filter PLL. BIOS controls this statically.		
42h	Extended Time Window	Tau for PL1		MSR 0x610 [23:17]
43h	Short Time Window	This is the time constant for PL2?	SNB-E and IVB-E only	MSR 0x610 [55:49]
44h	Secondary Memory VR Voltage	VR for the second band of DIMM	SNB-E, IVB-E, and HSW-E only	

ID	Name	Description	Platform Compatibility	Comments
45h	Reference Clock Ratio		SNB-E and IVB-E only	
46h	Vboot Voltage		SNB-E and IVB-E Only	
47h	Runtime Turbo Override Enable		SNB(-E) Only	
49h	Memory Clock Multiplier		IVB(-E) and later	
4Ah	Filter PLL Frequency		HSW Only	
4Bh	Dynamic SVID Control	Enable or disable SVID bus control for FIVR	HSW (-E) and BDW Only	OC MB IA 0x12/13 [31]
4Ch	Ring Ratio		HSW (-E) and later	OC MB CLR 0x10/11 [7:0]
4Dh	Ring Voltage Override		HSW (-E) and later	OC MB CLR 0x10/11 [19:8]
4Eh	Ring Voltage Mode		HSW (-E) and later	OC MB CLR 0x10/11 [20]
4Fh	Ring Voltage Offset		HSW (-E) and later	OC MB CLR 0x10/11 [31:21]
50h	Overclocking Enable	To lock/unlock system for overclocking		MSR 0x194 [20]
51h	Processor Graphics Slice Voltage Override		HSW and later	OC MB GTSLICE 0x10/11 [19:8]
52h	Processor Graphics Slice Voltage Mode		HSW and later	OC MB GTSLICE 0x10/11 [20]
53h	Processor Graphics Slice Voltage Offset		HSW and later	OC MB GTSLICE 0x10/11 [31:21]

ID	Name	Description	Platform Compatibility	Comments
54h	Package Current Limit		HSW(-E) and BDW only	MSR 0x601 [12:0] This is the same as the control 39h. Between IVB and HSW, the graphics current limit was dropped, so this was created to differentiate “Core Current Limit” from “Package Current Limit”
55h	System Agent Voltage Offset		HSW (-E) and later	OC MB SA 0x10/11 [31:21]
56h	FIVR Faults		HSW(-E) and BDW only	OC MB IA 0x14/15 [0]
57h	FIVR Efficiency Management		HSW (-E) and BDW only	OC MB IA 0x14/15 [1]
58h	IA Core Voltage Mode		HSW (-E) and later	OC MB IA 0x10/11 [20]
59h	SVID Voltage Override		HSW (-E) and BDW only	OC MB IA 0x12/13 [11:0]
5Ah	PEG/DMI Ratio	Use this ratio to bring PEG/DMI clock to a fixed 100Mhz when BCLK frequency changes.	HSW (-E) and BDW only	
5Bh	I/O Analog Voltage Offset	Voltage offset for analog I/O in the CPU package	HSW and BDW only	OC MB IOA 0x10/11 [31:21]
5Ch	I/O Digital Voltage Offset	Voltage offset for digital I/O in the CPU package	HSW and BDW only	OC MB IOD 0x10/11 [31:21]
5Dh	All Bank Row Pre-charge Delay Time (tRPab)			
5Eh	ED RAM PLL Ratio	The turbo ratio limit for eDRAM in the package.	BDW and later	MCHBAR 0x5E08
5Fh	Thermal Throttle Offset	Ability to thermal throttling offset at runtime.		MCHBAR 0x5E08
60h	7-Active Core Ratio Limit		HSW-E and later	MSR 0x1AD [55:48] – locked by OC lock bit (MSR 194h [20])
61h	8-Active Core Ratio Limit		HSW-E and later	MSR 0x1AD [63:56] – locked by OC lock bit (MSR 194h [20])

ID	Name	Description	Platform Compatibility	Comments
62h	Processor Graphics Unslice Voltage mode	To choose either adaptive or override mode for the processor graphics unslice power domain. With the adaptive mode, the supplied voltage increases linearly when the frequency increases. With the override mode, the supplied voltage is constant.	SKL and later	OC MB 0x10/11 [20], Domain ID = Graphics Unslice (3)
63h	Processor Graphics Unslice Voltage Override	If the override mode is selected, this is the requested voltage (mV) for the processor graphics unslice power domain.	SKL and later	OC MB 0x10/11 [19:8] , Domain ID = Graphics Unslice (3)
64h	Processor Graphics Unslice Voltage Offset	This is the range of voltage (+/- mV) that is allowed to be off the voltage target.	SKL and later	OC MB 0x10/11 [31:21] , Domain ID = Graphics Unslice (3)
65h	Processor Graphics Unslice Ratio Limit	This is the realtime ratio limit set by the software. It has to be lower than the fused ratio limit.	SKL and later	OC MB 0x10/11 [7:0] , Domain ID = Graphics Unslice (3)
66h	Processor Core Current Limit Maximum	This is the max current that is allowed for the CPU core voltage rail.	SKL and later	OC MB 0x4 IA VR address
67h	System Agent Current Limit Maximum	This is the max current that is allowed for the System Agent (a.k.a uncore) voltage rail.	SKL and later	OC MB 0x4 SA VR address
68h	Processor Graphics Slice Current Limit Maximum	This is the max current that is allowed for the processor graphics slice voltage rail.	SKL and later	OC MB 0x4 GTSLICE VR address
69h	Processor Graphics Unslice Current Limit Maximum	This is the max current that is allowed for the processor graphics unslice voltage rail.	SKL and later	OC MB 0x4 GTUnSlice VR address
6Ah	Ring Current Limit Maximum	The max current that Ring VR can support.	SKL and later	OC MB 0x4 Ring VR address

Table 31: Temperature (TSDD) Usage enumeration

Enumeration	Definition
00h	Unknown
01h	CPU Core
02h	CPU Die
05h	Voltage Regulator (VR)
06h	DIMM
07h	Motherboard Ambient
08h	System Ambient
09h	CPU Inlet
0Ah	System Inlet
0Bh	System Outlet
0Ch	Power Supply
0Dh	Power Supply Inlet
0Eh	Power Supply Outlet
0Fh	Hard Drive
10h	Graphics Processor Unit (GPU)
11h	Laptop Skin
12h	Optical Disk Drive
13h	PCMCIA slot
14h	PCH
15h	Battery

Table 32: Voltage (VSDD) Usage enumeration

Enumeration	Definition
00h	Unknown
01h	+12 Volt
02h	-12 Volt
03h	+5 Volt
04h	+5 Volt Backup
05h	-5 Volt
06h	3.3 Volt

Enumeration	Definition
07h	2.5 Volt
08h	1.5 Volt
09h	CPU Voltage
0Dh	Power Supply Inlet
0Fh	+3.3 Volt Standby
10h	CPU System Agent Voltage
11h	1.8 Volt
12h	PCH Voltage
13h	DDR Voltage
14h	Battery
15h	CPU IO Voltage
16h	CPU PLL Voltage

Table 33: Fan (FSDD) Usage enumeration

Enumeration	Definition
00h	Unknown/Other Usage
01h	CPU
02h	CPU System
04h	Voltage Regulator
05h	Chassis
06h	Chassis Inlet
07h	Chassis Outlet
08h	Power Supply
09h	Power Supply Inlet
0Ah	Power Supply Outlet
0Bh	Hard Disk
0Ch	Graphics
0Dh	Auxiliary
0Eh	PCH
0Fh	Battery
FFh	Unused