



NVIDIA WPT BIOS Guide

CONFIDENTIAL INFORMATION
DO NOT COPY

Version 2.02
December, 2008

A decorative graphic at the bottom of the page consisting of a green curved shape on the left and a blue and white wavy, metallic-looking shape on the right.

PLATFORM PROCESSORS

Document Change History

Version	Date	Owner	Reason for Change
0.1	03/01/05	D. Hunt	Initial version 0.1
0.2	07/15/05	D. Hunt	Updated Motherboard ID table for BFG, Clevo, J&W, Jiehe, ECS, and Paradise.
1.0	11/09/05	D. Hunt	Official release
1.1	01/13/06	D. Hunt	Updated Motherboard ID table for Dell, HP/Compaq, Gateway, and Winfast Removed "Generation 3.0" from title, since it is good for nTune 2.x forward.
1.5	04/19/06	D. Hunt	Added new functions for nTune 5.0 (Note: WPT version remains 2.0). Function 05h: GetNforceFanSpeed Function 09h: GetNforceFanPower Function 0Ah: SetNforceFanPower Function 0Bh: GetAux1FanSpeed Function 0Ch: GetAux1FanPower Function 0Dh: SetAux1FanPower Function 28h: GetAux2FanSpeed Function 29h: GetAux2FanPower Function 30h: SetAux2FanPower Function 15h: GetPSUTemp (Get power supply temp) Function 20h: GetHTCpuSppVoltage (HT for CPU & SPP) Function 21h: SetHTCpuSppVoltage Function 22h: GetHTSppMcpVoltage (HT for SPP & MCP) Function 23h: SetHTSppMcpVoltage
1.6	08/07/06	D. Hunt	Added new functions Function 35h: GetLEDInfo Function 36h: SetLEDInfo Corrected bit description for CPU VID Get/Set functions
1.7	09/01/06	D. Hunt	The function code of SetAux2FanPower should be 0x2A instead of 0x30.
1.8	02/13/08	D. Hunt	Corrected Processor descriptions in the voltage sections. Corrected HT Link descriptions Added Functions 2Ah and 2Bh for CPU GTL VRef
1.9	02/14/08	D. Hunt	Expanded CPU GTL VRef for 8 values.
1.95	2/21/08	D. Hunt	Added IDs for : "EVGA", "Alienware", "VoodooPC", "XFX"
2.00	07/29/08	M. Peters	Added new functions for AMD OC: Function 0x2Dh: GetACCStatus Function 0x2Eh: SetACCStatus
2.01	10/20/08	M. Peters	Added EIST bit to Get/Set CPU Voltage
2.02	12/18/08	M. Peters	Fixed typo in GTL Vref register listing

PROPRIETARY INFORMATION
DO NOT DISTRIBUTE

--	--	--	--

Table of Contents

1.0	BIOS Interface	1
1.1	Description of Communication	1
1.1.1	General Description	1
1.1.1.1	Status Codes	2
1.2	Sub-functions	2
1.2.1	Version and Board Identification	2
1.2.1.1	NVIDIA WPT BIOS Interface Version	2
1.2.1.2	Motherboard Information	3
1.2.1.3	Read BIOS Information	2
1.2.2	Fan Control	3
1.2.2.1	CPU Fan	3
1.2.2.2	System Fan	4
1.2.2.3	NVIDIA nForce Fan	5
1.2.2.4	NVIDIA Auxiliary 1 Fan	6
1.2.2.5	NVIDIA Auxiliary 2 Fan	7
1.2.3	Voltage Control	8
1.2.3.1	Returning Precise Voltage Levels	8
1.2.3.2	CPU Voltage	9
1.2.3.3	Chipset Core or MCP Core Voltage	10
1.2.3.4	External AGP, PCI-E or SPP Core Voltage	11
1.2.3.5	Memory Voltage	12
1.2.3.6	Auxiliary Voltage	13
1.2.3.7	HT Link (AMD) / FSB (Intel) for SPP to CPU Voltage	14
1.2.3.8	HT LINK for SPP to MCP Voltage	15
1.2.3.9	CPU GTL VRef Voltage	16
1.2.4	Temperature Reporting	17
1.2.4.1	CPU Temperature	17
1.2.4.2	System Temperature	17
1.2.4.3	Additional CPU Temperature	17
1.2.4.4	Power Supply Unit Temperature	18
1.2.5	CMOS Manipulation	19
1.2.5.1	Checksum Calculation	19
1.2.6	Flash Update Control	19
1.2.6.1	Bootblock Control	19
1.2.6.2	Flash Access	20
1.2.7	LED Management	21
1.2.7.1	Get LED colors/effects	21
1.2.7.2	Set LED colors/effects	21
1.2.8	CPU Management	22
1.2.8.1	Advanced Clock Calibration	22
2.0	Sample Code	23
2.1	Tables	23
2.1.1	Function Table For The SMI Handler	23

**PROPRIETARY INFORMATION
DO NOT DISTRIBUTE**

2.1.2	Voltage Tables.....	24
2.1.2.1	Chipset Core Voltage Table.....	24
2.1.2.2	AGP Voltage Table.....	24
2.1.2.3	Memory Voltage Table.....	25
2.1.2.4	Auxiliary Voltage Table.....	25
2.1.3	Voltage GPIO Control Tables.....	26
2.1.3.1	DefaultVoltageMarginTable.....	26
2.1.3.2	CPU Voltage Margin GPIO Table.....	28
2.2	Entry and Data Handling.....	29
2.2.1	Defining The SMI Function Code.....	29
2.2.2	Main SMI Entry.....	29
2.2.3	Data Handling.....	31
2.2.3.1	Get Input Data.....	31
2.2.3.2	Set Return Data.....	31
2.3	Support Functions.....	32
2.3.1	DefaultFunc:.....	32
2.3.2	GetInterfaceVersion:.....	32
2.3.3	EnableHWMonitor.....	33
2.3.4	CS_I2CReadByte.....	34
2.3.5	CS_I2CwriteByte.....	35
2.3.6	CS_Chk_SMBus_READY.....	36
2.3.7	SuperIORegRead.....	37
2.3.8	SuperIORegWrite.....	37
2.4	Temperature Functions.....	38
2.4.1	GetCPUtemperature.....	38
2.4.2	GetMotherBoardTemperature.....	39
2.5	Voltage Functions.....	40
2.5.1	Voltage Support Functions.....	40
2.5.1.1	EnableVoltageMargin.....	40
2.5.1.2	EnableDefaultVoltMargin.....	41
2.5.1.3	GetVoltMarginOutenStatus.....	42
2.5.1.4	EnableVIDOverrideCtrl.....	43
2.5.1.5	GetVIDOverrideCtrlStatus.....	44
2.5.1.6	SetOverrideVoltage.....	45
2.5.1.7	GetOverrideGPIOVoltage.....	46
2.5.1.8	FindVoltageInTable.....	47
2.5.2	Auxiliary Voltage.....	48
2.5.2.1	GetAuxVoltage.....	48
2.5.2.2	SetAuxVoltage.....	49
2.5.3	Memory Voltage.....	50
2.5.3.1	GetMemoryVoltage.....	50
2.5.3.2	SetMemoryVoltage.....	51
2.5.4	AGP Voltage.....	52
2.5.4.1	GetAGPVoltage.....	52
2.5.4.2	SetAGPVoltage.....	53
2.5.5	Chipset Core Voltage.....	54
2.5.5.1	GetCSCoreVoltage.....	54
2.5.5.2	SetCSCoreVoltage.....	55
2.5.6	CPU Voltage.....	56
2.5.6.1	SetCPUVID.....	56
2.5.6.2	GetCPUVID.....	57

PROPRIETARY INFORMATION
DO NOT DISTRIBUTE

2.6	Fan Functions	58
2.6.1	CPU Fan Functions.....	58
2.6.1.1	GetCPUFanSpeed	58
2.6.1.2	GetCPUFanPower	60
2.6.1.3	SetCPUFanPower	61
2.6.2	System Fan Functions	62
2.6.2.1	GetSecondFanSpeed	62
2.6.2.2	GetSecondFanPower	64
2.6.2.3	SetSecondFanPower	65
2.7	CMOS Functions	66
2.7.1	WPTComputeCMOSCheckSumAll	66
2.7.2	WPTComputeCMOSCheckSum.....	67
2.7.3	Motherboard Information	68
2.7.3.1	ReadMBInfo	68
2.7.3.2	GetBoardVersion	69
2.7.3.3	Read BIOS Information	69
2.8	Source Definitions.....	70

1.0 BIOS Interface

To enable/customized certain features of the NVIDIA System Monitor, NVIDIA System Update, and NVIDIA Performance Control Panel; the System BIOS must provide a set of functions in the SMI handler. To invoke these functions, driver will generate a software SMI with a function code defined by the OEM in the MCP Scratch Register 7 bit(s) [31:24]. This should be defined at the end of POST.

Note: For the purpose of the provided examples, C8h is used for the SMI function code.

During POST, system control registers of the Media Communications Processor are mapped to an IO base address that can be read from bus 0, device 1, function 0, offset 64h-65h. To ensure software SMI is enabled, registers offset 05h bit2 and offset 04h bit0 needs to be 1. In general, these bits should already be set by BIOS during POST.

Note: For the purpose of the provided examples, the System Control IO base is assumed to be 4400h.

1.1 Description of Communication

1.1.1 General Description

Writing function code C8h to register offset 2Eh will generate software SMI related to NVIDIA SW events. Register offset 2Fh will be used for input sub-function code and output status code. Currently, 01h to 27h are reserved for these sub-functions. After a SMI is handled, status code SUCCESS, INVALID_FUNCTION, or FAILURE will be returned in register-offset 2Fh. The driver will check register offset 2Fh for status each time after a SMI is generated and handled. Scratch registers offset 84h-8Fh are used to pass data in and out during SMI handling. Among these scratch registers, offset 88h to 8Fh are always used to pass a description string to driver when an OEM GET function is called. The Driver will save the value of these scratch registers before using them and restore them after the SMI is handled.



NVIDIA.

NVIDIA Corporation
2701 San Tomas Expressway
Santa Clara, CA 95050
www.nvidia.com

1.1.1.1 Status Codes

Status codes returned in register offset 2Fh:

Return	Description
00h	SUCCESS: Sub-function execution succeeds.
FFh	INVALID_FUNCTION: Returned if sub-function code is not within range of 01h to 2Eh.
FEh	FAILURE: Sub-function execution fails or is not supported.

1.2 Sub-functions

1.2.1 Version and Board Identification

1.2.1.1 NVIDIA WPT BIOS Interface Version

1.2.1.1.1 Sub-function code 01h: Get the NVIDIA WPT interface version.

Code	Description
Out 442Fh, 01h	Sub-function code
Out 442Eh, 0C8h	Software SMI function code
In al, 442Fh	Either SUCCESS, INVALID_FUNCTIONCODE, or FAILURE will be returned in register 442Fh.
In ax, 4484h	Returned interface version will be saved in register 4484h and 4485h. Register 4485h is major version and Register 4484h is minor version. So 0200h means version 2.0

The most significant word of this sub-function is used as an indication for OEM function support. If it is read as 0001h, the OEM function is supported. Otherwise, no OEM function support from BIOS. Corresponding code change in BIOS is as following:

```
OEMSUPPORT      equ      00 ;Not supported
in      sub function 01:
mov     ecx, (OEMSUPPORT shl 16) + WPT_INTERFACE_VERSION
```

**PROPRIETARY INFORMATION
DO NOT DISTRIBUTE**

1.2.1.2 Motherboard Information

1.2.1.2.1 Sub-function code 25h

Code	Description
Out 442Fh, 25h	Sub-function code
Out 442Eh, 0C8h	Software SMI function code
In al, 442Fh	Either SUCCESS, INVALID_FUNCTIONCODE, or FAILURE will be returned in register 442Fh.

The motherboard vendor ID is returned in MCP Scratch register 1 bits [15:0] (see Motherboard Vendor ID table). Bits [31:16] of the register are used to reflect the board version (ex. 0100h = v1.0).

Motherboard Vendor ID Table

ID Number	Vendor Name	ID Number	Vendor Name
0x00	Not Supported	0x16	SOYO
0x01	NVIDIA	0x17	Arima
0x02	Asus	0x18	ECS
0x03	MSI	0x19	Foxconn
0x04	Abit	0x1A	Tyan
0x05	Chaintech	0x1B	TCL
0x06	Epox	0x1C	Top Star
0x07	Leadtek	0x1D	Huawei
0x08	Shuttle	0x1E	BFG
0x09	Gigabyte	0x1F	Clevo
0x0A	FIC	0x20	J&W
0x0B	Aopen	0x21	Jiehe
0x0C	Soltek	0x22	ECS
0x0D	Biostar	0x23	Paradise
0x0E	Wistron	0x24	Dell
0x0F	Mitac	0x25	HP/Compaq
0x10	DFI	0x26	Gateway
0x11	Albatron	0x27	Winfast
0x12	Jetway	0x28	EVGA
0x13	Surwin	0x29	Alienware
0x14	QDI/Legend	0x2A	VoodooPC
0x15	Pine	0x2B	XFX

The MCP Scratch registers 2 through 7 will be used for the Mother board model. Each byte represents the Hex equivalent of an ASCII character. Any register bytes not used should be populated with 0.

1.2.1.3 Read BIOS Information

This is used by the BIOS update dialog in NVIDIA System Update and the NVIDIA Performance Control Panel. If it is not supported, the information is shown as not reported.

1.2.1.3.1 Sub-function code 27h

Code	Description
Out 442Fh, 27h	Sub-function code
Out 442Eh, 0C8h	Software SMI function code
In al, 442Fh	Either SUCCESS, INVALID_FUNCTIONCODE, or FAILURE will be returned in register 442Fh.

Data will be saved in Scratch registers 1-7.

Register	Bit(s)	Description
Scratch Reg 1	31:16	BIOS version (ex. 0100h = v1.0)
Scratch Reg 1	15:0	BIOS Vendor ID (see: Motherboard Vendor ID table under “Motherboard Information”)
Scratch Regs 3:2	31:0	Each byte represents the Hex equivalent of an ASCII character. Any register bytes not used should be populated with 0.
Scratch Reg 4	31:16 15:8 7:0	BIOS date – year (as a number) BIOS date – month (as a number) BIOS date – day (as a number)
Scratch Regs 7(23:0): 5	xx:0	BIOS ID (total 11 Chars). If BIOS_ID is less than 11 characters long, fill the rest of register with 0. This will be used to block flashing the wrong BIOS on the wrong board.
Scratch reg 7	31:24	CMOS version- Version can be set between 1-255 (0xFF). This will be used to disable the “Access BIOS” page if the CMOS map version differs from the nvsuoem.ini setting.

**PROPRIETARY INFORMATION
DO NOT DISTRIBUTE**

1.2.2 Fan Control

1.2.2.1 CPU Fan

1.2.2.1.1 Sub-function code 02h: Get CPU Fan Speed

Code	Description
Out 442Fh, 02h	Sub-function code
Out 442Eh, 0C8h	Software SMI function code
In al, 442Fh	Either SUCCESS, INVALID_FUNCTIONCODE, or FAILURE will be returned in register 442Fh.
In eax, 4484h	Returned fan speed in RPM as an DWORD will be saved in register 4484h.
In eax, 4488h	Returned description string.
In edx, 4489h	Returned description string.

1.2.2.1.2 Sub-function code 03h: Get CPU Fan Power

Code	Description
Out 442Fh, 03h	Sub-function code
Out 442Eh, 0C8h	Software SMI function code
In al, 442Fh	Either SUCCESS, INVALID_FUNCTIONCODE, or FAILURE will be returned in register 442Fh.
In al, 4484h	Returned fan power will be saved in register 4484h. Value will be one byte between 00 to FF. FF means full power on.
In eax, 4488h	Returned description string.
In edx, 4489h	Returned description string.

1.2.2.1.3 Sub-function code 04h: Set CPU Fan Power

Code	Description
Out 442Fh, 04h	Sub-function code
Out 4484h, al	Fan power Value between 00 to FF is given to 4484h.
Out 442Eh, 0C8h	Software SMI function code
In al, 442Fh	Either SUCCESS, INVALID_FUNCTIONCODE, or FAILURE will be returned in register 442Fh.

1.2.2.2 System Fan

1.2.2.2.1 Sub-function code 06h: Get System Fan Speed

Code	Description
Out 442Fh, 06h	Sub-function code
Out 442Eh, 0C8h	Software SMI function code
In al, 442Fh	Either SUCCESS, INVALID_FUNCTIONCODE, or FAILURE will be returned in register 442Fh.
In eax, 4484h	Returned fan speed in RPM as an DWORD will be saved in register 4484h.
In eax, 4488h	Returned description string.
In edx, 4489h	Returned description string.

1.2.2.2.2 Sub-function code 07h: Get System Fan Power

Code	Description
Out 442Fh, 07h	Sub-function code
Out 442Eh, 0C8h	Software SMI function code
In al, 442Fh	Either SUCCESS, INVALID_FUNCTIONCODE, or FAILURE will be returned in register 442Fh.
In al, 4484h	Returned fan power will be saved in register 4484h. Value will be one byte between 00 to FF. FF means full power on.
In eax, 4488h	Returned description string.
In edx, 4489h	Returned description string.

1.2.2.2.3 Sub-function code 08h: Set System Fan Power

Code	Description
Out 442Fh, 08h	Sub-function code
Out 4484h, al	Fan power Value between 00 to FF is given to 4484h.
Out 442Eh, 0C8h	Software SMI function code
In al, 442Fh	Either SUCCESS, INVALID_FUNCTIONCODE, or FAILURE will be returned in register 442Fh.

**PROPRIETARY INFORMATION
DO NOT DISTRIBUTE**

1.2.2.3 NVIDIA nForce Fan

1.2.2.3.1 Sub-function code 05h: Get NVIDIA nForce Fan Speed

Code	Description
Out 442Fh, 05h	Sub-function code
Out 442Eh, 0C8h	Software SMI function code
In al, 442Fh	Either SUCCESS, INVALID_FUNCTIONCODE, or FAILURE will be returned in register 442Fh.
In eax, 4484h	Returned fan speed in RPM as an DWORD will be saved in register 4484h.
In eax, 4488h	Returned description string.
In edx, 4489h	Returned description string.

1.2.2.3.2 Sub-function code 09h: Get NVIDIA nForce Fan Power

Code	Description
Out 442Fh, 09h	Sub-function code
Out 442Eh, 0C8h	Software SMI function code
In al, 442Fh	Either SUCCESS, INVALID_FUNCTIONCODE, or FAILURE will be returned in register 442Fh.
In al, 4484h	Returned fan power will be saved in register 4484h. Value will be one byte between 00 to FF. FF means full power on.
In eax, 4488h	Returned description string.
In edx, 4489h	Returned description string.

1.2.2.3.3 Sub-function code 0Ah: Set NVIDIA nForce Fan Power

Code	Description
Out 442Fh, 0Ah	Sub-function code
Out 4484h, al	Fan power Value between 00 to FF is given to 4484h.
Out 442Eh, 0C8h	Software SMI function code
In al, 442Fh	Either SUCCESS, INVALID_FUNCTIONCODE, or FAILURE will be returned in register 442Fh.

1.2.2.4 NVIDIA Auxiliary 1 Fan

1.2.2.4.1 Sub-function code 0Bh: Get Auxiliary 1 Fan Speed

Code	Description
Out 442Fh, 0Bh	Sub-function code
Out 442Eh, 0C8h	Software SMI function code
In al, 442Fh	Either SUCCESS, INVALID_FUNCTIONCODE, or FAILURE will be returned in register 442Fh.
In eax, 4484h	Returned fan speed in RPM as an DWORD will be saved in register 4484h.
In eax, 4488h	Returned description string.
In edx, 4489h	Returned description string.

1.2.2.4.2 Sub-function code 0Ch: Get Auxiliary 1 Fan Power

Code	Description
Out 442Fh, 0Ch	Sub-function code
Out 442Eh, 0C8h	Software SMI function code
In al, 442Fh	Either SUCCESS, INVALID_FUNCTIONCODE, or FAILURE will be returned in register 442Fh.
In al, 4484h	Returned fan power will be saved in register 4484h. Value will be one byte between 00 to FF. FF means full power on.
In eax, 4488h	Returned description string.
In edx, 4489h	Returned description string.

1.2.2.4.3 Sub-function code 0Dh: Set Auxiliary 1 Fan Power

Code	Description
Out 442Fh, 0Dh	Sub-function code
Out 4484h, al	Fan power Value between 00 to FF is given to 4484h.
Out 442Eh, 0C8h	Software SMI function code
In al, 442Fh	Either SUCCESS, INVALID_FUNCTIONCODE, or FAILURE will be returned in register 442Fh.

**PROPRIETARY INFORMATION
DO NOT DISTRIBUTE**

1.2.2.5 NVIDIA Auxiliary 2 Fan

1.2.2.5.1 Sub-function code 28h: Get Auxiliary 2 Fan Speed

Code	Description
Out 442Fh, 28h	Sub-function code
Out 442Eh, 0C8h	Software SMI function code
In al, 442Fh	Either SUCCESS, INVALID_FUNCTIONCODE, or FAILURE will be returned in register 442Fh.
In eax, 4484h	Returned fan speed in RPM as an DWORD will be saved in register 4484h.
In eax, 4488h	Returned description string.
In edx, 4489h	Returned description string.

1.2.2.5.2 Sub-function code 29h: Get Auxiliary 2 Fan Power

Code	Description
Out 442Fh, 29h	Sub-function code
Out 442Eh, 0C8h	Software SMI function code
In al, 442Fh	Either SUCCESS, INVALID_FUNCTIONCODE, or FAILURE will be returned in register 442Fh.
In al, 4484h	Returned fan power will be saved in register 4484h. Value will be one byte between 00 to FF. FF means full power on.
In eax, 4488h	Returned description string.
In edx, 4489h	Returned description string.

1.2.2.5.3 Sub-function code 2Ah: Set Auxiliary 2 Fan Power

Code	Description
Out 442Fh, 2Ah	Sub-function code
Out 4484h, al	Fan power Value between 00 to FF is given to 4484h.
Out 442Eh, 0C8h	Software SMI function code
In al, 442Fh	Either SUCCESS, INVALID_FUNCTIONCODE, or FAILURE will be returned in register 442Fh.

1.2.3 Voltage Control

1.2.3.1 Returning Precise Voltage Levels

1.2.3.1.1 Sub-function code 24h

Purpose: Get Actual Voltage Reading

Code	Description
Out 442Fh, 24h	Sub-function code
Out 442Eh, 0C8h	Software SMI function code
In al, 442Fh	Either SUCCESS, INVALID_FUNCTIONCODE, or FAILURE will be returned in register 442Fh.

Data will be returned in integer format (voltage * 1000000) in the following MCP Scratch registers.

Scratch Register	Format	Data Returned
1	dword	CPU actual voltage reading
2	dword	Chipset or MCP core actual voltage reading
3	dword	AGP, PCIE or SPP core actual voltage reading
4	dword	Memory actual voltage reading
5	dword	Auxiliary actual voltage reading
6	dword	HT CPU-SPP actual voltage reading
7	dword	HT SPP-MCP actual voltage reading

IMPORTANT: The System BIOS needs to return actual voltage for FSB.

Example:

If the voltage is set to 1.2v and offset is -0.1v, the voltage returned in this call is 1.1v.

1.2.3.2 CPU Voltage

1.2.3.2.1 Sub-function code 16h: Get CPU Voltage

Code	Description
Out 442Fh, 16h	Sub-function code
Out 442Eh, 0C8h	Software SMI function code
In al, 442Fh	Either SUCCESS, INVALID_FUNCTIONCODE, or FAILURE will be returned in register 442Fh.
In eax, 4484h	<p>Returned voltage will be saved in register 4484h. The format is using the Standard CPU VID.</p> <p>Byte0 [7:0]= Standard CPU VID (Refer to appropriate CPU specification)</p> <p>Byte1 [15:8] extends the Maximum voltage above the CPU VID reported xx000001 = Max + (1*0.025) or Max + 0.025 xx000010 = Max + (2*0.025) or Max + 0.05 xx000011 = Max + (3*0.025) or Max + 0.075</p> <p>Byte2 [16] EIST enabled (Available in 6.03.07.00 or later)</p>

1.2.3.2.2 Sub-function code 17h: Set CPU Voltage

Code	Description
Out 442Fh, 17h	Sub-function code
Out 4484, eax	Voltage is given as defined by VID in the Get example.
Out 442Eh, 0C8h	Software SMI function code
In al, 442Fh	Either SUCCESS, INVALID_FUNCTIONCODE, or FAILURE will be returned in register 442Fh.

Some System BIOSes may not be able to switch from manual voltage control to EIST controlled so the actual voltage should still be indicated through the Set function.

1.2.3.3 Chipset Core or MCP Core Voltage

1.2.3.3.1 Sub-function code 18h: Get Chipset Core Voltage

Code	Description
Out 442Fh, 18h	Sub-function code
Out 442Eh, 0C8h	Software SMI function code
In al, 442Fh	Either SUCCESS, INVALID_FUNCTIONCODE, or FAILURE will be returned in register 442Fh.
In eax, 4484h	Returned voltage will be saved in registers from 4484h to 4487h. The format is integer format. Ex. 1.7 is 1700000 and .8 is 800000 and .025 is 25000.
In eax, 4488h	Returned description string.
In edx, 4489h	Returned description string.

1.2.3.3.2 Sub-function code 19h: Set Chipset Core Voltage

Code	Description
Out 442Fh, 19h	Sub-function code
Out 4484, eax	Voltage is given in integer format (voltage * 1000000).
Out 442Eh, 0C8h	Software SMI function code
In al, 442Fh	Either SUCCESS, INVALID_FUNCTIONCODE, or FAILURE will be returned in register 442Fh.

**PROPRIETARY INFORMATION
DO NOT DISTRIBUTE**

1.2.3.4 External AGP, PCI-E or SPP Core Voltage

1.2.3.4.1 Sub-function code 1Ah: Get AGP Voltage

Code	Description
Out 442Fh, 1Ah	Sub-function code
Out 442Eh, 0C8h	Software SMI function code
In al, 442Fh	Either SUCCESS, INVALID_FUNCTIONCODE, or FAILURE will be returned in register 442Fh.
In eax, 4484h	Returned voltage will be saved in registers from 4484h to 4487h. The format is integer format. Ex. 1.7 is 1700000 and .8 is 800000 and .025 is 25000.
In eax, 4488h	Returned description string.
In edx, 4489h	Returned description string.

1.2.3.4.2 Sub-function code 1Bh: Set AGP Voltage

Code	Description
Out 442Fh, 1Bh	Sub-function code
Out 4484, eax	Voltage is given in integer format (voltage * 1000000).
Out 442Eh, 0C8h	Software SMI function code
In al, 442Fh	Either SUCCESS, INVALID_FUNCTIONCODE, or FAILURE will be returned in register 442Fh.

1.2.3.5 Memory Voltage

1.2.3.5.1 Sub-function code 1Ch: Get Memory Voltage

Code	Description
Out 442Fh, 1Ch	Sub-function code
Out 442Eh, 0C8h	Software SMI function code
In al, 442Fh	Either SUCCESS, INVALID_FUNCTIONCODE, or FAILURE will be returned in register 442Fh.
In eax, 4484h	Returned voltage will be saved in registers from 4484h to 4487h. The format is integer format. Ex. 1.7 is 1700000 and .8 is 800000 and .025 is 25000.
In eax, 4488h	Returned description string.
In edx, 4489h	Returned description string.

1.2.3.5.2 Sub-function code 1Dh: Set Memory Voltage

Code	Description
Out 442Fh, 1Dh	Sub-function code
Out 4484, eax	Voltage is given in integer format (voltage * 1000000).
Out 442Eh, 0C8h	Software SMI function code
In al, 442Fh	Either SUCCESS, INVALID_FUNCTIONCODE, or FAILURE will be returned in register 442Fh.

**PROPRIETARY INFORMATION
DO NOT DISTRIBUTE**

1.2.3.6 Auxiliary Voltage

1.2.3.6.1 Sub-function 1Eh: Get AUX Voltage

Code	Description
Out 442Fh, 1Eh	Sub-function code
Out 442Eh, 0C8h	Software SMI function code
In al, 442Fh	Either SUCCESS, INVALID_FUNCTIONCODE, or FAILURE will be returned in register 442Fh.
In eax, 4484h	Returned voltage will be saved in registers from 4484h to 4487h. The format is integer format. Ex. 1.7 is 1700000 and .8 is 800000 and .025 is 25000.
In eax, 4488h	Returned description string.
In edx, 4489h	Returned description string.

1.2.3.6.2 Sub-function code 1Fh: Set AUX Voltage

Code	Description
Out 442Fh, 1Fh	Sub-function code
Out 4484, eax	Voltage is given in integer format (voltage * 1000000).
Out 442Eh, 0C8h	Software SMI function code
In al, 442Fh	Either SUCCESS, INVALID_FUNCTIONCODE, or FAILURE will be returned in register 442Fh.

1.2.3.7 HT Link (AMD) / FSB (Intel) for SPP to CPU Voltage

1.2.3.7.1 Sub-function 20h: Get HT Link / FSB for SPP to CPU Voltage

Code	Description
Out 442Fh, 20h	Sub-function code
Out 442Eh, 0C8h	Software SMI function code
In al, 442Fh	Either SUCCESS, INVALID_FUNCTIONCODE, or FAILURE will be returned in register 442Fh.
In eax, 4484h	<p>Returned voltage will be saved in registers from 4484h to 4487h. The format is integer format. Ex. 1.7 is 1700000 and .8 is 800000 and .025 is 25000.</p> <p>Scratch Register 0: Has the value to offset the voltage returned in scratch register 1. If no offset is required, use 0 here.</p> <p>Scratch Register 1: Voltage returned needs to match the ones defined in INI file.</p> <p>Example:</p> <p>Current FSB voltage is 1.2 and offset is -0.1. So 1200000 is returned in scratch register 1 and -100000 is returned in scratch register 0 as signed value.</p>
In eax, 4488h	Returned description string.
In edx, 4489h	Returned description string.

1.2.3.7.2 Sub-function code 21h: Set HT Link / FSB for SPP to CPU Voltage

Code	Description
Out 442Fh, 21h	Sub-function code
Out 4484, eax	<p>Voltage is given in integer format (voltage * 1000000).</p> <p>Example:</p> <p>If Performance wants to set 1.2v, scratch register 1 will be set to 1200000 before calling SMI.</p>
Out 442Eh, 0C8h	Software SMI function code

**PROPRIETARY INFORMATION
DO NOT DISTRIBUTE**

1.2.3.8 HT LINK for SPP to MCP Voltage

1.2.3.8.1 Sub-function 22h: Get HT LINK for SPP to MCP Voltage

Code	Description
Out 442Fh, 22h	Sub-function code
Out 442Eh, 0C8h	Software SMI function code
In al, 442Fh	Either SUCCESS, INVALID_FUNCTIONCODE, or FAILURE will be returned in register 442Fh.
In eax, 4484h	Returned voltage will be saved in registers from 4484h to 4487h. The format is integer format. Ex. 1.7 is 1700000 and .8 is 800000 and .025 is 25000.
In eax, 4488h	Returned description string.
In edx, 4489h	Returned description string.

1.2.3.8.2 Sub-function code 23h: Set HT LINK for SPP to MCP Voltage

Code	Description
Out 442Fh, 23h	Sub-function code
Out 4484, eax	Voltage is given in integer format (voltage * 1000000).
Out 442Eh, 0C8h	Software SMI function code
In al, 442Fh	Either SUCCESS, INVALID_FUNCTIONCODE, or FAILURE will be returned in register 442Fh.

1.2.3.9 CPU GTL VRef Voltage

1.2.3.9.1 Sub-function code 2Bh: Get CPU GTL VRef Voltage

Code	Description
Out 442Fh, 2Bh	Sub-function code
Out 442Eh, 0C8h	Software SMI function code
In al, 442Fh	Either SUCCESS, INVALID_FUNCTIONCODE, or FAILURE will be returned in register 442Fh.
In eax, 4480h	Scratch register offset 0x80-0x9F for CPU GTL VRef voltage, 0x88-0x8B for CPU GTL VRef 1 voltage 0x8C-0x8F for CPU GTL VRef 2 voltage 0x90-0x93 for CPU GTL VRef 3 voltage 0x94-0x97 for CPU GTL VRef 4 voltage 0x98-0x9B for CPU GTL VRef 5 voltage 0x9C-0x9F for CPU GTL VRef 6 voltage 0x80-0x83 for CPU GTL VRef 7 voltage 0x84-0x87 for CPU GTL VRef 8 voltage Note: 0xFFFFFFFF means not supported. Note: The returned value is a signed value.

1.2.3.9.2 Sub-function code 2Ch: Set CPU GTL VRef Voltage

Code	Description
Out 442Fh, 2Ch	Sub-function code
Out 4480, eax	Scratch register offset 0x80-0x9F for CPU GTL VRef voltage. 0x88-0x8B for CPU GTL VRef 1 voltage 0x8C-0x8F for CPU GTL VRef 2 voltage 0x90-0x93 for CPU GTL VRef 3 voltage 0x94-0x97 for CPU GTL VRef 4 voltage 0x98-0x9B for CPU GTL VRef 5 voltage 0x9C-0x9F for CPU GTL VRef 6 voltage 0x80-0x83 for CPU GTL VRef 7 voltage 0x84-0x87 for CPU GTL VRef 8 voltage Note: The input value is a signed value.
Out 442Eh, 0C8h	Software SMI function code
In al, 442Fh	Either SUCCESS, INVALID_FUNCTIONCODE, or FAILURE will be returned in register 442Fh.

1.2.4 Temperature Reporting

1.2.4.1 CPU Temperature

1.2.4.1.1 Sub-function code 12h: Get CPU Temperature

Code	Description
Out 442Fh, 12h	Sub-function code
Out 442Eh, 0C8h	Software SMI function code
In al, 442Fh	Either SUCCESS, INVALID_FUNCTIONCODE, or FAILURE will be returned in register 442Fh.

Data is returned in three bytes.

Byte0 = Temperature as one signed byte in Celsius.

Byte1 = Temperature threshold for over-temp condition.

Byte2 = Bit 0: 0 = CPU Die temperature reported, 1 = CPU Case temperature reported.

1.2.4.2 System Temperature

1.2.4.2.1 Sub-function code 13h: Get Motherboard Temperature

Code	Description
Out 442Fh, 13h	Sub-function code
Out 442Eh, 0C8h	Software SMI function code
In al, 442Fh	Either SUCCESS, INVALID_FUNCTIONCODE, or FAILURE will be returned in register 442Fh.

Data is returned in three bytes.

Byte0 = Temperature as one signed byte in Celsius.

Byte1 = Temperature threshold for over-temp condition.

1.2.4.3 Additional CPU Temperature

1.2.4.3.1 Sub-function code 14h: Get 2nd CPU Temperature

Code	Description
Out 442Fh, 14h	Sub-function code
Out 442Eh, 0C8h	Software SMI function code
In al, 442Fh	Either SUCCESS, INVALID_FUNCTIONCODE, or FAILURE will be returned in register 442Fh.

See: CPU Temperature for returned data format.

1.2.4.4 Power Supply Unit Temperature

This is for a non-ESA implementation of power supply temperature monitoring.

1.2.4.4.1 Sub-function code 15h: Get Power Supply Unit Temperature

Code	Description
Out 442Fh, 15h	Sub-function code
Out 442Eh, 0C8h	Software SMI function code
In al, 442Fh	Either SUCCESS, INVALID_FUNCTIONCODE, or FAILURE will be returned in register 442Fh.

Data is returned in three bytes.

Byte0 = Temperature as one signed byte in Celsius.

Byte1 = Temperature threshold for over-temp condition.

1.2.5 CMOS Manipulation

When Control Panel updates settings from the “Dynamic BIOS Acces” page, it is necessary to correct the CMOS checksum after the CMOS values are changed.

1.2.5.1 Checksum Calculation

1.2.5.1.1 Sub-function 26h: Recalculate CMOS checksum.

Code	Description
Out 442Fh, 26h	Sub-function code
Out 442Eh, 0C8h	Software SMI function code
In al, 442Fh	Either SUCCESS, INVALID_FUNCTIONCODE, or FAILURE will be returned in register 442Fh.

1.2.6 Flash Update Control

1.2.6.1 Bootblock Control

1.2.6.1.1 Sub-function 0Eh: Enable Bootblock Write

Code	Description
Out 442Fh, 0Eh	Sub-function code
Out 442Eh, 0C8h	Software SMI function code
In al, 442Fh	Either SUCCESS, INVALID_FUNCTIONCODE, or FAILURE will be returned in register 442Fh.

This call should enable write access to flash ROM bootblock. NVIDIA System Update will only call this function when bootblock update is requested by the user.

1.2.6.1.2 Sub-function 0Fh: Disable Bootblock Write

Code	Description
Out 442Fh, 0Fh	Sub-function code
Out 442Eh, 0C8h	Software SMI function code
In al, 442Fh	Either SUCCESS, INVALID_FUNCTIONCODE, or FAILURE will be returned in register 442Fh.

This call should disable write access to flash ROM bootblock. NVIDIA System Update will only call this function after the bootblock update is completed.

1.2.6.2 Flash Access

1.2.6.2.1 Sub-function 10h: Enable Flash Access

Code	Description
Out 442Fh, 10h	Sub-function code
Out 442Eh, 0C8h	Software SMI function code
In al, 442Fh	Either SUCCESS, INVALID_FUNCTIONCODE, or FAILURE will be returned in register 442Fh.

This call should enable write access to the flash ROM.

1.2.6.2.2 Sub-function 11h: Disable Flash Access

Code	Description
Out 442Fh, 11h	Sub-function code
Out 442Eh, 0C8h	Software SMI function code
In al, 442Fh	Either SUCCESS, INVALID_FUNCTIONCODE, or FAILURE will be returned in register 442Fh.

This call should disable write access to the flash ROM.

**PROPRIETARY INFORMATION
DO NOT DISTRIBUTE**

1.2.7 LED Management

This is only for use with non-ESA LED management solutions

1.2.7.1 Get LED colors/effects

1.2.7.1.1 Sub-function 35h: Get LED colors/effects

Code	Description
Out 442Fh, 35h	Sub-function code
Out 442Eh, 0C8h	Software SMI function code
In al, 442Fh	Either SUCCESS, INVALID_FUNCTIONCODE, or FAILURE will be returned in register 442Fh.
In ax, 4484h	<p>Returned colors and features will be saved in register 4484h.</p> <p>LED color [3:0] As defined in the INI file</p> <p>Features [6:4] Defines the supported features</p> <p>Not supported 000 Chase 011 Fade 010</p> <p>Blink 001 Pendulum 100</p> <p>Zone [11:8] Defines the zone number</p> <p>Intensity [15:12] Defines the intensity level (0000b is MIN)</p>

1.2.7.2 Set LED colors/effects

1.2.7.2.1 Sub-function 36h: Set LED colors/effects

Code	Description
Out 442Fh, 36h	Sub-function code
Out ax, 4484h	<p>Colors and features will be given in register 4484h.</p> <p>LED color [3:0] As defined in the INI file</p> <p>Features [6:4] Apply feature</p> <p>Not supported 000 Chase 011 Fade 010</p> <p>Blink 001 Pendulum 100</p> <p>Zone [11:8] To this zone number</p> <p>Intensity [15:12] Set intensity level (0000b is MIN)</p>
Out 442Eh, 0C8h	Software SMI function code
In al, 442Fh	Either SUCCESS, INVALID_FUNCTIONCODE, or FAILURE will be returned in register 442Fh.

1.2.8 CPU Management

The CPU management WPT functions are currently only supported on system boards containing AMD processors which can be communicated with through the JTAG via the SMU.

1.2.8.1 Advanced Clock Calibration

1.2.8.1.1 Sub-function 2Dh: GetACCStatus

Code	Description
Out 442Fh, 2Dh	Sub-function code
Out 442Eh, 0C8h	Software SMI function code
In al, 442Fh	Either SUCCESS, INVALID_FUNCTIONCODE, or FAILURE will be returned in register 442Fh.
In eax, 4484h	Read in Scratch register data from 84h, 88h, and 8Ch (see below)

Data will be saved in Scratch registers 1-3.

Register	Bit(s)	Description
Scratch Reg 1	31:16	Reserved
	15:8	Number of cores supporting ACC
	7:2	Reserved
	1:0	0 Disabled 1 Auto 2 All cores 3 Per core
Scratch Reg 2	31:24	Value of ACC for Core 4 if supported
	23:16	Value of ACC for Core 3
	15:8	Value of ACC for Core 2
	7:0	Value of ACC for Core 1
Scratch Reg 3	31:24	Value of ACC for Core 8 if supported
	23:16	Value of ACC for Core 7 if supported
	15:8	Value of ACC for Core 6 if supported
	7:0	Value of ACC for Core 5 if supported

1.2.8.1.2 Sub-function 2Eh: SetACCStatus

Code	Description
Out 442Fh, 2Eh	Sub-function code
Out 4484h, eax	Above scratch register data to 84h, 88h, and 8Ch
Out 442Eh, 0C8h	Software SMI function code
In al, 442Fh	Either SUCCESS, INVALID_FUNCTIONCODE, or FAILURE will be returned in register 442Fh.

2.0 Sample Code

The Sample code provided is written for the NVIDIA Reference board (PCB 180-7R012-0000-D00); therefore, it may require changes dependent upon the specific OEM hardware design.

2.1 Tables

2.1.1 Function Table For The SMI Handler

public WPTFunctionTable WPTFunctionTable:		
	;Function Offset	Function code
dw	(offset CS:GetInterfaceVersion - offset CS:SMI_Handler_Start)	;01h
dw	(offset CS:GetCPUFanSpeed - offset CS:SMI_Handler_Start)	;02h
dw	(offset CS:GetCPUFanPower - offset CS:SMI_Handler_Start)	;03h
dw	(offset CS:SetCPUFanPower - offset CS:SMI_Handler_Start)	;04h
dw	(offset CS:DefaultFunc - offset CS:SMI_Handler_Start)	;05h
dw	(offset CS:GetSecondFanSpeed - offset CS:SMI_Handler_Start)	;06h
dw	(offset CS:GetSecondFanPower - offset CS:SMI_Handler_Start)	;07h
dw	(offset CS:SetSecondFanPower - offset CS:SMI_Handler_Start)	;08h
dw	(offset CS:DefaultFunc - offset CS:SMI_Handler_Start)	;09h
dw	(offset CS:DefaultFunc - offset CS:SMI_Handler_Start)	;0Ah
dw	(offset CS:DefaultFunc - offset CS:SMI_Handler_Start)	;0Bh
dw	(offset CS:DefaultFunc - offset CS:SMI_Handler_Start)	;0Ch
dw	(offset CS:DefaultFunc - offset CS:SMI_Handler_Start)	;0Dh
dw	(offset CS:EnableBootblockWrite - offset CS:SMI_Handler_Start)	;0Eh
dw	(offset CS:DisableBootblockWrite - offset CS:SMI_Handler_Start)	;0Fh
dw	(offset CS:EnableFlashRomAccess - offset CS:SMI_Handler_Start)	;10h
dw	(offset CS:DisableFlashRomAccess - offset CS:SMI_Handler_Start)	;11h
dw	(offset CS:GetCPUtemperature - offset CS:SMI_Handler_Start)	;12h
dw	(offset CS:GetMotherBoardTemperature - offset CS:SMI_Handler_Start)	;13h
dw	(offset CS:GetSecondCPUtemperature - offset CS:SMI_Handler_Start)	;14h
dw	(offset CS:DefaultFunc - offset CS:SMI_Handler_Start)	;15h
dw	(offset CS:GetCPUVID - offset CS:SMI_Handler_Start)	;16h
dw	(offset CS:SetCPUVID - offset CS:SMI_Handler_Start)	;17h
dw	(offset CS:GetCSCoreVoltage - offset CS:SMI_Handler_Start)	;18h
dw	(offset CS:SetCSCoreVoltage - offset CS:SMI_Handler_Start)	;19h

**PROPRIETARY INFORMATION
DO NOT DISTRIBUTE**

```

dw  (offset CS:GetAGPVoltage - offset CS:SMI_Handler_Start)      ;1Ah
dw  (offset CS:SetAGPVoltage - offset CS:SMI_Handler_Start)      ;1Bh
dw  (offset CS:GetMemoryVoltage - offset CS:SMI_Handler_Start)    ;1Ch
dw  (offset CS:SetMemoryVoltage - offset CS:SMI_Handler_Start)    ;1Dh
dw  (offset CS:GetAuxVoltage - offset CS:SMI_Handler_Start)       ;1Eh
dw  (offset CS:SetAuxVoltage - offset CS:SMI_Handler_Start)       ;1Fh
dw  (offset CS:DefaultFunc - offset CS:SMI_Handler_Start)        ;20h
dw  (offset CS:DefaultFunc - offset CS:SMI_Handler_Start)        ;21h
dw  (offset CS:DefaultFunc - offset CS:SMI_Handler_Start)        ;22h
dw  (offset CS:DefaultFunc - offset CS:SMI_Handler_Start)        ;23h
dw  (offset CS:ReadActualVoltage - offset CS:SMI_Handler_Start)   ;24h
dw  (offset CS:ReadMBInfo - offset CS:SMI_Handler_Start)         ;25h
dw  (offset CS:WPTComputeCMOSChecksumAll - offset CS:SMI_Handler_Start) ;26h
dw  (offset CS:ReadBIOSInfo - offset CS:SMI_Handler_Start)       ;27h
dw  (offset CS:DefaultFunc - offset CS:SMI_Handler_Start)        ;28h
dw  (offset CS:DefaultFunc - offset CS:SMI_Handler_Start)        ;29h
dw  (offset CS:DefaultFunc - offset CS:SMI_Handler_Start)        ;2Ah

```

WPTFunctionTableLen EQU (\$ - WPTFunctionTable)

2.1.2 Voltage Tables

2.1.2.1 Chipset Core Voltage Table

```

public CSCoreVoltageTable
CSCoreVoltageTable:
CSCoreVoltageTable:
dd  1700000      ;1.70v      00b
dd  1600000      ;1.60v      01b
dd  1600000      ;1.60v      02b
dd  1500000 ;Default ;1.50v    03b
dd  VOLTAGE_TABLE_END ;End OF table

```

2.1.2.2 AGP Voltage Table

```

public AGPVoltageTable
AGPVoltageTable:
dd  1700000      ;1.70v      00b
dd  1600000      ;1.60v      01b
dd  1600000      ;1.60v      02b
dd  1500000      ;Default;1.50v    03b
dd  VOLTAGE_TABLE_END ;End OF table

```

2.1.2.3 Memory Voltage Table

```
public MemoryVoltageTable
MemoryVoltageTable:
    dd    2800000          ;2.80v      00b
    dd    2700000          ;2.70v      01b
    dd    2700000          ;2.70v      02b
    dd    2600000          ;Default;2.60v  03b
    dd    VOLTAGE_TABLE_END ;End OF table
```

2.1.2.4 Auxiliary Voltage Table

```
public AuxVoltageTable
AuxVoltageTable: ;Default is 1.5v
    dd    1600000          ;1.60v      00b
    dd    1700000          ;1.70v      01b
    dd    1700000          ;1.70v      02b
    dd    00000000h         ;NULL      03b
    dd    VOLTAGE_TABLE_END ;End OF table
```

2.1.3 Voltage GPIO Control Tables

For the examples provided, the GPIO assignments are for a SMSC LPC47M157-NC SIO controller as configured on the NVIDIA C17-SPP and C18D reference platforms.

2.1.3.1 DefaultVoltageMarginTable

The table description is for each GPIO pin that controls voltage margin.

Note: the following table will be within the following table definition.

```
public DefaultVoltageMarginTable
DefaultVoltageMarginTable:
; AuxVoltageGPIONTable:
; MemoryVoltageGPIONTable:
; AGPVoltageGPIONTable:
; CSCoreVoltageGPIONTable:
DefaultVoltageMarginTableLen EQU ($ - DefaultVoltageMarginTable)
```

2.1.3.1.1 Auxiliary Voltage GPIO Table

AuxVoltageGPIONTable:

```
;Aux Margin1
db AUX_MARGIN1_GPIO12_CTRL ; GPIO control register
db GPIO_CTRL_OUTPUT ; Set to out function
db AUX_GPIO_GROUP ; GPIO group data register
db AUX_MARGIN1_DEFAULT_STATUS ; Default Bit status. Only can be 0 and 1.
db AUX_MARGIN1_BIT ; Bit number for this bit in the GPIO group data
```

reg.

```
GPIONTableSingleSetLen EQU ($ - AuxVoltageGPIONTable)
```

```
;Aux Margin2
db AUX_MARGIN2_GPIO17_CTRL
db GPIO_CTRL_OUTPUT
db AUX_GPIO_GROUP
db AUX_MARGIN2_DEFAULT_STATUS
db AUX_MARGIN2_BIT
```

```
AuxVoltageGPIONTableLen EQU ($ - AuxVoltageGPIONTable)
```

PROPRIETARY INFORMATION
DO NOT DISTRIBUTE

2.1.3.1.2 Memory Voltage GPIO Table

MemoryVoltageGPIONTable:

```
;Memory margin1
db  MEM_MARGIN1_GPIO15_CTRL
db  GPIO_CTRL_OUTPUT
db  MEM_GPIO_GROUP
db  MEM_MARGIN1_DEFAULT_STATUS
db  MEM_MARGIN1_BIT
;Memory Margin2
db  MEM_MARGIN2_GPIO16_CTRL
db  GPIO_CTRL_OUTPUT
db  MEM_GPIO_GROUP
db  MEM_MARGIN2_DEFAULT_STATUS
db  MEM_MARGIN2_BIT
```

MemoryVoltageGPIONTableLen EQU (\$ - MemoryVoltageGPIONTable)

2.1.3.1.3 AGP Voltage GPIO Table

AGPVoltageGPIONTable:

```
;AGP Margin1
db  AGP_MARGIN1_GPIO13_CTRL
db  GPIO_CTRL_OUTPUT
db  AGP_GPIO_GROUP
db  AGP_MARGIN1_DEFAULT_STATUS
db  AGP_MARGIN1_BIT
;AGP Margin2
db  AGP_MARGIN2_GPIO14_CTRL
db  GPIO_CTRL_OUTPUT
db  AGP_GPIO_GROUP
db  AGP_MARGIN2_DEFAULT_STATUS
db  AGP_MARGIN2_BIT
```

AGPVoltageGPIONTableLen EQU (\$ - AGPVoltageGPIONTable)

2.1.3.1.4 Chipset Core Voltage GPIO Table

CSCoreVoltageGPIONTable:

```
;Chipset core Margin1
db  CSCORE_MARGIN1_GPIO10_CTRL
db  GPIO_CTRL_OUTPUT
db  CSCORE_GPIO_GROUP
db  CSCORE_MARGIN1_DEFAULT_STATUS
db  CSCORE_MARGIN1_BIT
;Chipset Core Margin2
db  CSCORE_MARGIN2_GPIO11_CTRL
db  GPIO_CTRL_OUTPUT
db  CSCORE_GPIO_GROUP
```

```
db    CSCORE_MARGIN2_DEFAULT_STATUS
db    CSCORE_MARGIN2_BIT
CSCoreVoltageGPIONTableLen    EQU    ($ - CSCoreVoltageGPIONTable)
db    VOLTARGINEN_GPIO43_CTRL
db    GPIO_CTRL_OUTPUT
db    VOLTARGINEN_GPIO_GROUP
db    MARGINOUTEN_ENABLE_STATUS
db    MARGINOUTEN_ENABLE_BIT
```

2.1.3.2 CPU Voltage Margin GPIO Table

CPUVoltageGPIONTable:

```
;VID0
db    CPUVID_OVERRIDE0_GPIO25_CTRL
db    GPIO_VID_CTRL_OUTPUT
db    CPUVID_GPIO_GROUP
db    CPUVID_DEFAULT_STATUS
db    CPUVID0_GPIO_BIT
;VID1
db    CPUVID_OVERRIDE1_GPIO26_CTRL
db    GPIO_VID_CTRL_OUTPUT
db    CPUVID_GPIO_GROUP
db    CPUVID_DEFAULT_STATUS
db    CPUVID1_GPIO_BIT
;VID2
db    CPUVID_OVERRIDE2_GPIO20_CTRL
db    GPIO_VID_CTRL_OUTPUT
db    CPUVID_GPIO_GROUP
db    CPUVID_DEFAULT_STATUS
db    CPUVID2_GPIO_BIT
;VID3
db    CPUVID_OVERRIDE3_GPIO21_CTRL
db    GPIO_VID_CTRL_OUTPUT
db    CPUVID_GPIO_GROUP
db    CPUVID_DEFAULT_STATUS
db    CPUVID3_GPIO_BIT
;VID4
db    CPUVID_OVERRIDE4_GPIO22_CTRL
db    GPIO_VID_CTRL_OUTPUT
db    CPUVID_GPIO_GROUP
db    CPUVID_DEFAULT_STATUS
db    CPUVID4_GPIO_BIT
```

CPUVoltageGPIONTableLen EQU (\$ - CPUVoltageGPIONTable)

2.2 Entry and Data Handling

2.2.1 Defining The SMI Function Code

This macro should be used in the end of POST to set correct function code. The NVIDIA application will get the function code from the MCP Scratch Register 7 bit(s) [31:24] and use it to access the WPT functions.

```
SetWPTFuncCode MACRO          ;This macro should be used when stack is available.
    push    dx
    push    ax
    mov     dx, SMI_IO_BASE_ADDR + 9Fh    ;Scratch register 7 bit 24-31.
    mov     al, NV_WPT
    out     dx, al
    pop     ax
    pop     dx
ENDM
```

2.2.2 Main SMI Entry

This is the main entry for the WPT interface. Check if it is a WPT request and if it is, perform the proper function and return the value. Data is updated to AL and ECX in SMM state save area.

```
WPT_SMI_Entry_Main proc near
    push    ax
    push    bx
    push    ecx

    ; Read input function code and saved in al. Read input data and saved in ecx
    call    GetInputData
    cmp     al, 00h
    je      WPT_SMI_Entry_Main_Invalid    ;invalid function code
    cmp     al, (WPTFunctionTableLen shr 1)
    ja      WPT_SMI_Entry_Main_Invalid    ;invalid function code
    mov     bx, (offset CS:WPTFunctionTable - offset CS:SMI_Handler_Start)
    xor     ah, ah
    dec     al                            ;Calculate correct offset for function
    shl     al, 1
    add     bx, ax
    mov     bx, cs:[bx]
    call    bx                            ; return value in ecx
    jc      WPT_SMI_Entry_Main_Execute_Fail ; carry set means failed.
    mov     al, WPT_RETURN_SUCCESS
```

```
    jmp      WPT_SMI_Entry_Main_Exit

WPT_SMI_Entry_Main_Execute_Fail:
    mov     al, WPT_RETURN_FAIL
    jmp     WPT_SMI_Entry_Main_Exit
WPT_SMI_Entry_Main_Invalid:
    mov     al, WPT_RETURN_INVALID
WPT_SMI_Entry_Main_Exit:
    call    SetReturnData

    pop     ecx
    pop     bx
    pop     ax
    ret

WPT_SMI_Entry_Main endp
```

CONFIDENTIAL INFORMATION
DO NOT COPY

2.2.3 Data Handling

2.2.3.1 Get Input Data

Output: AL has the function code and ECX has the data.

GetInputData proc near

```
push    dx

mov     dx, SMI_IO_BASE_ADDR+SW_SMI_SCRATCH1_REG
in      eax, dx
mov     ecx, eax
mov     dx, SMI_IO_BASE_ADDR+SW_SMI_STATUS_REG
in      al, dx

pop     dx
ret
```

GetInputData endp

2.2.3.2 Set Return Data

Input: AL has the return code and ECX has the return value; Output: AL and ECX in SMM state save table are updated.

SetReturnData proc near

```
push    dx
push    si
push    es

mov     dx, SMI_IO_BASE_ADDR+SW_SMI_STATUS_REG
out     dx, al
mov     dx, SMI_IO_BASE_ADDR+SW_SMI_SCRATCH1_REG
mov     eax, ecx
out     dx, eax

pop     es
pop     si
pop     dx
ret
```

SetReturnData endp

2.3 Support Functions

2.3.1 DefaultFunc:

Output: jump to WPT_SMI_Entry_Main_Invalid label

DefaultFunc proc near

```
WPTRestoreStack  
jmp     WPT_SMI_Entry_Main_Invalid
```

DefaultFunc endp

2.3.2 GetInterfaceVersion:

Output: data is saved in ECX. Carry clear means success; else failure

GetInterfaceVersion proc near

```
mov     ecx, WPT_INTERFACE_VERSION  
  
clc  
ret
```

GetInterfaceVersion endp

2.3.3 EnableHWMonitor

Output: data is saved in ECX. Carry clear means success; else failure

This is used by the temperature and voltage control functions.

EnableHWMonitor proc near

```
    push    ax
    push    bx

    mov     bl, HW_MONITOR_SMBUS_READ_ADDR
    mov     al, SMSC_LPC47M15_SMB_REG_CONFIGURATION
    call    CS_I2cReadByte
    jc      EnableHWMonitorExit

    test    bh, 1
    jnz     EnableHWMonitorExit    ;jump if HWMonitor already enabled.

    or      bh, 21h
    mov     bl, HW_MONITOR_SMBUS_WRITE_ADDR
    mov     al, SMSC_LPC47M15_SMB_REG_CONFIGURATION
    call    CS_I2CWriteByte
```

EnableHWMonitorExit:

```
    pop     bx
    pop     ax
    ret
```

EnableHWMonitor endp

2.3.4 CS_I2CReadByte

This is used only for SMM mode and is used by the temperature and voltage control functions.
Input: BL - 8 bits I2C ID address, AL - byte index. Output: BH - value return if carry cleared, else failed

CS_I2CReadByte Proc Near

```

    push    dx

    ; Disable stuff in extended control reg.
    push    ax
    mov     al, 00
    mov     dx, SMBUS_PORT2 + 3Eh
    out     dx, al

    ; Send a not in use to protocol reg.
    mov     dx, SMBUS_PORT2 + 00h
    out     dx, al

    pop     ax

    mov     ah, al          ; byte index
    mov     al, bl
    mov     dx, SMBUS_PORT2 + 02h
    out     dx, ax          ; address and index
    ; wait SMBus ready
    ; ROM_CALL Chk_SMBus_READY ;N18
    ; start I2C read operation
    newiodelay
    mov     dx, SMBUS_PORT2 + 00h
    mov     al, 07h          ; MCP_SMB_PRTCL_PROTOCOL_READ_BYTE
    out     dx, al
    newiodelay
    ; check status OK ?
    call    CS_Chk_SMBus_READY
    jc      CS_I2C_ReadFail    ; SMBus Fail
    newiodelay
    ; read data
    mov     dx, SMBUS_PORT2 + 04h ; MCP_CONST_SMB1_BASE +
MCP_SMB_DATA_00
    in      al, dx
    mov     bh, al          ; value return
CS_I2C_ReadFail:

    pop     dx

```

PROPRIETARY INFORMATION
DO NOT DISTRIBUTE

```
ret
CS_I2CReadByte Endp
```

2.3.5 CS_I2CwriteByte

This is used only for SMM mode and is used by the temperature and voltage control functions.
Input: BL - 8 bits I2C ID address, AL - byte index, BH - value to write. Output: Fail - if carry set.

```
CS_I2CWriteByte Proc Near
    push    dx

    mov     ah, al        ; byte index
    mov     al, bl
    mov     dx, SMBUS_PORT2 + 02h
    out     dx, ax        ; address and index
    newiodelay
    ; put data to Smbus Base + 6
    mov     al, bh
    mov     dx, SMBUS_PORT2 + 04h
    out     dx, al        ; Data0
    newiodelay
    ; wait Smbus ready
    call    CS_Chk_SMBus_READY
    ; start I2C write operation
    newiodelay
    mov     dx, SMBUS_PORT2 + 00h
    mov     al, 06h        ; MCP_SMB_PRTCL_PROTOCOL_WRITE_BYTE
    out     dx, al
    ; check status OK ?
    call    CS_Chk_SMBus_READY

    pop     dx
    ret
CS_I2CWriteByte Endp
```

2.3.6 CS_Chk_SMBus_READY

This is used only for SMM mode. Output: Carry set means error and carry clear means ready.

```
CS_Chk_SMBus_READY Proc Near
    push    dx
    push    cx
    push    ax
    mov     dx, SMBUS_PORT2 + 01h    ; Status port
    cld
```

```
    mov     cx, 2000h
CS_Chk_I2C_OK:
    newiodelay
    in      al, dx                    ; get status
    or      al, al                    ; N18
    jz      CS_Chk_I2C_OK             ; not ready

    test    al, 80h                   ; is finished
    jnz     CS_Clear_final

    test    al, 01Fh
    jz      CS_Clear_final

    and     al, 01Fh
    cmp     al, 01Ah                  ; SMBus busy?
    jne     CS_SMBus_Err

    loop    CS_Chk_I2C_OK
```

CS_SMBus_Err:

```
    stc
CS_Clear_final:
    out     dx, al                    ; clear status
    pop     ax
    pop     cx
    pop     dx
    ret
CS_Chk_SMBus_READY Endp
```

2.3.7 SuperIORegRead

Read Super IO registers. Entry: AH: Register Index. Exit: AL: the register value read.

```
SuperIORegRead proc near
    push    dx

    mov     dx, SUPERIO_GPIO_PORT_BASE
    add     dl, ah
    in      al, dx

    pop     dx
    ret
SuperIORegRead endp
```

2.3.8 SuperIORegWrite

Read Super IO register. Entry: AH: Register Index. AL: Register value

```
SuperIORegWrite proc near
    push    dx

    mov     dx, SUPERIO_GPIO_PORT_BASE
    add     dl, ah
    out     dx, al

    pop     dx
    ret
SuperIORegWrite endp
```

2.4 Temperature Functions

Note: For the temperature monitoring, the following part must be removed from the reference board (PCB 180-7R012-0000-D00): U12G2.

2.4.1 GetCPUTemperature

Reads the CPU die temperature from the CPU internal temperature sensor. Output: data is saved in ECX. Carry clear means success, else failure.

GetCPUTemperature proc near

```
    push    ax

    call    EnableHWMonitor
    jc      GetCPUTemperatureError
    mov     bl, HW_MONITOR_SMBUS_READ_ADDR
    mov     al, CPU_TEMP_SMBUS_OFFSET
    call    CS_I2cReadByte
    jc      GetCPUTemperatureError

    movzx   ecx, bh ;Save current temperature in ecx
    mov     ch, CPU_TEMP_THRESHOLD
    cld
```

GetCPUTemperatureError:

```
    pop     ax
    ret
```

GetCPUTemperature endp

2.4.2 GetMotherBoardTemperature

Reads a temperature sensor connected to the SIO on the motherboard. Output: data is saved in ECX. Carry clear means success, else failure.

GetMotherBoardTemperature proc near

```
    push    ax

    call    EnableHWMonitor
    jc      GetMBTemperatureError

    mov     bl, HW_MONITOR_SMBUS_READ_ADDR
    mov     al, BOARD_TEMP_SMBUS_OFFSET
    call    CS_I2cReadByte
    jc      GetMBTemperatureError

    movzx   ecx, bh
    mov     ch, MB_TEMP_THRESHOLD
    cld
```

GetMBTemperatureError:

```
    pop     ax
    ret
```

GetMotherBoardTemperature endp

2.5 Voltage Functions

2.5.1 Voltage Support Functions

2.5.1.1 EnableVoltageMargin

Default table loaded.

Entry: SI: table length, DI: table offset, DL: GPIO override status (Bit 0 = 1: override with new data, Bit 0 = 0: don't override), DH: bit 0 is the data for GPIO override if override is enabled. Exit: DI: is in the end of the table.

EnableVoltageMargin proc near

```
    push    ax
    push    cx
```

```
    ; Set GPIO to proper status
```

SetNextGPIOReg:

```
    mov     ah, cs:[di]
    mov     al, cs:[di+1]
    call    SuperIORegWrite
```

```
    mov     ah, cs:[di+2]
    call    SuperIORegRead
    mov     cx, cs:[di+3]
```

```
    xchg    ch, cl
    ror     al, cl
    and     al, 0FEh
    rol     al, cl
    xchg    ch, cl
```

```
    test    dl, 1                ;Check for override
    jz      NoOverrideVoltage
    mov     cl, ch
    shl     dh, cl
    mov     cl, dh
```

NoOverrideVoltage:

```
    or      al, cl

    call    SuperIORegWrite
```

**PROPRIETARY INFORMATION
DO NOT DISTRIBUTE**

```

add     di, GPIOTableSingleSetLen
sub     si, GPIOTableSingleSetLen
cmp     si, 0
jne     SetNextGPIOReg

pop     cx
pop     ax
ret
EnableVoltageMargin endp

```

2.5.1.2 EnableDefaultVoltMargin

EnableDefaultVoltMargin proc near

```

push    si
push    di
push    dx

xor     dx, dx                ;no override for default data
mov     si, DefaultVoltageMarginTableLen
mov     di, (offset CS:DefaultVoltageMarginTable - offset CS:SMI_Handler_Start)
;find voltage in table
call    EnableVoltageMargin

clc
pop     dx
pop     di
pop     si

ret
EnableDefaultVoltMargin endp

```

2.5.1.3 GetVoltMarginOutenStatus

Check status of Volt_margin_out_en. Exit: carry set if enabled (override enabled)

GetVoltMarginOutenStatus proc near

```
    push    ax
```

```
    ;check if GPIO as output and GPIO function.
```

```
    mov     ah, VOLTARGINEN_GPIO43_CTRL
```

```
    call    SuperIORegRead
```

```
    and     al, GPIO_CTRL_OUTPUT_MASK
```

```
    cmp     al, 0
```

```
    jne     MarginNotEnabled
```

```
    mov     ah, VOLTARGINEN_GPIO_GROUP
```

```
    call    SuperIORegRead
```

```
    test    al, VOLTARGINEN_GPIOMASK
```

```
    jz      MarginNotEnabled
```

```
    stc
```

```
    jmp     GetVoltageStatusExit
```

MarginNotEnabled:

```
    clc
```

GetVoltageStatusExit:

```
    pop     ax
```

```
    ret
```

GetVoltMarginOutenStatus endp

2.5.1.4 EnableVIDOverrideCtrl

Enable Volt_Margin_Outen

EnableVIDOverrideCtrl proc near

push ax

;Enable gpio as output and gpio function.

mov ah, VID_OVERRIDE_GPIO55_CTRL

mov al, GPIO_CTRL_OUTPUT

call SuperIORegWrite

mov ah, VID_OVERRIDE_GPIO_GROUP

call SuperIORegRead

or al, VID_OVERRIDE_GPIOMASK

call SuperIORegWrite

pop ax

ret

EnableVIDOverrideCtrl endp

2.5.1.5 GetVIDOverrideCtrlStatus

Check status of VID_OVERRIDE_CTRL. Exit: Carry set if enabled (override enabled)

GetVIDOverrideCtrlStatus proc near

push ax

;check if gpio as output and gpio function.

mov ah, VID_OVERRIDE_GPIO55_CTRL

call SuperIORegRead

and al, GPIO_CTRL_OUTPUT_MASK

cmp al, 0

jne MarginNotEnabled

mov ah, VID_OVERRIDE_GPIO_GROUP

call SuperIORegRead

test al, VID_OVERRIDE_GPIOMASK

jz VIDOverrideNotEnabled

stc

jmp GetVIDOverrideStatusExit

VIDOverrideNotEnabled:

clc

GetVIDOverrideStatusExit:

pop ax

ret

GetVIDOverrideCtrlStatus endp

2.5.1.6 SetOverrideVoltage

Entry: DI: offset that GPIO table start, SI: Table length, al: data to set. Exit: Carry set means failure.

SetOverrideVoltage proc near

```
    push    dx
    push    cx
```

```
    mov     cx, si                ; cx used make a loop for how many bit is to set.
```

SetNextOverrideVolMargin:

```
    mov     dl, 1                ; Override with data in dh
```

```
    mov     dh, al
```

```
    and     dh, 01h              ; bit 0 for Margin 1
```

```
    mov     si, GPIOTableSingleSetLen
```

```
    ;When this loop run on second time
```

```
    ;di has the value point to next entry of the table.
```

```
    call    EnableVoltageMargin
```

```
    sub     cx, GPIOTableSingleSetLen
```

```
    shr     al, 1
```

```
    cmp     cx, 0
```

```
    jne     SetNextOverrideVolMargin
```

```
    pop     cx
```

```
    pop     dx
```

```
    ret
```

SetOverrideVoltage endp

2.5.1.7 GetOverrideGPIOVoltage

Entry: DI: offset that GPIO table start, SI: Table length. Exit: if carry clear, AX has the data.
Carry set means GPIO is not used to control voltage.

GetOverrideGPIOVoltage proc near

```

    push    bx
    push    cx

    xor     bx, bx
GetGPIOVolNextTBLSet:
    mov     ah, cs:[di]
    call    SuperIORegRead
    cmp     al, cs:[di+1]
    jne     GetGPIOVoltageNotEnable
    mov     ah, cs:[di+2]
    call    SuperIORegRead
    mov     cx, cs:[di+3]
    xchg    ch, cl
    shr     al, cl
    rcr     al, 1           ;shift bit 0 to carry
    rcr     bl, 1           ;shift carry to bl bit 7
    add     di, GPIONTableSingleSetLen
    inc     bh
    sub     si, GPIONTableSingleSetLen
    cmp     si, 0
    je      GetGPIOVoltageEnabled
    jmp     GetGPIOVolNextTBLSet
GetGPIOVoltageNotEnable:
    stc
    jmp     GetGPIOVoltageExit
GetGPIOVoltageEnabled:
    ; At this point, Data is in bl from bit 7 to 6.
    ; bh has the count how many rcr is done.
    ; compute how many shr still left to be done.
    mov     al, 8
    sub     al, bh
    mov     cl, al
    shr     bl, cl
    movzx   ax, bl    ;al has the data.
    clc
GetGPIOVoltageExit:
    pop     cx
    pop     bx
    ret
GetOverrideGPIOVoltage endp

```

2.5.1.8 FindVoltageInTable

Entry: ECX has the Voltage value; DI: has the Voltage Table offset. Exit: AL: return the corresponding entry in the table. Carry set means failure.

FindVoltageInTable proc near

```
    push    dx
    ;find voltage in table
    xor     al, al
```

FindVoltageAgain:

```
    cmp     ecx, cs:[di]
    je      GotVoltageInTable
    add     di, 4
    inc     al
    mov     dl, cs:[di+3]
    test    dl, 80h                ;voltage won't be negative.
    jnz     FindVoltageFail
    jmp     FindVoltageAgain
```

FindVoltageFail:

```
    stc
    jmp     FindVoltageExit
```

GotVoltageInTable:

```
    cld
```

FindVoltageExit:

```
    pop     dx
    ret
```

FindVoltageInTable endp

2.5.2 Auxiliary Voltage

2.5.2.1 GetAuxVoltage

Output: data is saved in ECX. Carry clear means success, else failure.

GetAuxVoltage proc near

```
    push    ax
    push    bx
```

```
    call    GetVoltMarginOutenStatus
    jnc     GetStandardAuxVolt
```

```
    mov     si, AuxVoltageGPIOTableLen
    mov     di, (offset CS:AuxVoltageGPIOTable - offset CS:SMI_Handler_Start)
;find voltage in table
    call    GetOverrideGPIOVoltage
    jc      GetStandardAuxVolt
```

```
    mov     bx, (offset CS:AuxVoltageTable - offset CS:SMI_Handler_Start)
;find voltage in table
    shl     ax, 2
    add     bx, ax
    mov     ecx, cs:[bx]    ;Set voltage to ecx
```

```
    jmp     AuxVoltConvert
```

GetStandardAuxVolt:

```
    mov     ecx, AUX_DEFAULT_VOLTAGE_1500    ;Give default voltage to ecx
```

AuxVoltConvert:

```
    cld
    pop     bx
    pop     ax
    ret
```

GetAuxVoltage endp

2.5.2.2 SetAuxVoltage

Entry: ECX has the value to set. Exit: Carry set means failure.

SetAuxVoltage proc near

```
    push    ax
    push    si
    push    di
    ;find voltage in table
    mov     di, (offset CS:AuxVoltageTable - offset CS:SMI_Handler_Start)
;find voltage in table
    call    FindVoltageInTable
    jc      SetAuxVoltageExit      ;Voltage not found then exit.

    call    GetVoltMarginOutenStatus
    jc      SetAuxVoltageMarginEnabled

    call    EnableDefaultVoltMargin

SetAuxVoltageMarginEnabled:
    mov     di, (offset CS:AuxVoltageGPIOTable - offset CS:SMI_Handler_Start)
;find voltage in table
    mov     si, AuxVoltageGPIOTableLen
    ;data in al already
    call    SetOverrideVoltage

    cld

SetAuxVoltageExit:
    pop     di
    pop     si
    pop     ax
    ret
SetAuxVoltage endp
```

2.5.3 Memory Voltage

2.5.3.1 GetMemoryVoltage

Output: data is saved in ECX. Carry clear means success, else failure.

GetMemoryVoltage proc near

```
    push    ax
    push    bx

    call    GetVoltMarginOutenStatus
    jnc     GetStandardMemoryVolt

    mov     si, MemoryVoltageGPIONTableLen
    mov     di, (offset CS:MemoryVoltageGPIONTable - offset CS:SMI_Handler_Start)
;find voltage in table
    call    GetOverrideGPIOVoltage
    jc      GetStandardMemoryVolt

    mov     bx, (offset CS:MemoryVoltageTable - offset CS:SMI_Handler_Start)
;find voltage in table
    shl     ax, 2
    add     bx, ax
    mov     ecx, cs:[bx]          ;Set voltage to ecx

    jmp     MemoryVoltConvert
GetStandardMemoryVolt:
    mov     ecx, MEM_DEFAULT_VOLTAGE_2600
MemoryVoltConvert:

    clc
    pop     bx
    pop     ax
    ret
GetMemoryVoltage endp
```

**PROPRIETARY INFORMATION
DO NOT DISTRIBUTE**

2.5.3.2 SetMemoryVoltage

Entry: ECX has the value to set. Exit: Carry set means failure.

SetMemoryVoltage proc near

```

    push    ax
    push    si
    push    di
    ;find voltage in table
    mov     di, (offset CS:MemoryVoltageTable - offset CS:SMI_Handler_Start)
;find voltage in table
    call    FindVoltageInTable
    jc      SetMemVoltageExit

    call    GetVoltMarginOutenStatus
    jc      SetMemVoltageMarginEnabled

    call    EnableDefaultVoltMargin

SetMemVoltageMarginEnabled:
    mov     di, (offset CS:MemoryVoltageGPIOTable - offset CS:SMI_Handler_Start)
;find voltage in table
    mov     si, MemoryVoltageGPIOTableLen
    ;data in al already
    call    SetOverrideVoltage

    cld

SetMemVoltageExit:
    pop     di
    pop     si
    pop     ax
    ret

SetMemoryVoltage endp

```

2.5.4 AGP Voltage

2.5.4.1 GetAGPVoltage

Output: data is saved in ECX. Carry clear means success, else failure.

GetAGPVoltage proc near

```
    push    ax
    push    bx

    call    GetVoltMarginOutenStatus
    jnc     GetStandardAGPVolt

    mov     si, AGPVoltageGPIOTableLen
    mov     di, (offset CS:AGPVoltageGPIOTable - offset CS:SMI_Handler_Start)
;find voltage in table
    call    GetOverrideGPIOVoltage
    jc      GetStandardAGPVolt

    mov     bx, (offset CS:AGPVoltageTable - offset CS:SMI_Handler_Start)
;find voltage in table
    shl     ax, 2
    add     bx, ax
    mov     ecx, cs:[bx]          ;Set voltage to ecx

    jmp     AGPVoltConvert
GetStandardAGPVolt:
    mov     ecx, AGP_DEFAULT_VOLTAGE_1500
AGPVoltConvert:

    clc
    pop     bx
    pop     ax
    ret
GetAGPVoltage endp
```

2.5.4.2 SetAGPVoltage

Entry: ECX has the value to set. Exit: Carry set means failure.

SetAGPVoltage proc near

push ax

push si

push di

;find voltage in table

mov di, (offset CS:AGPVoltageTable - offset CS:SMI_Handler_Start)

;find voltage in table

call FindVoltageInTable

jc SetAGPVoltageExit

call GetVltMarginOutenStatus

jc SetAGPVoltageMarginEnabled

call EnableDefaultVltMargin

SetAGPVoltageMarginEnabled:

mov di, (offset CS:AGPVoltageGPIOTable - offset CS:SMI_Handler_Start)

;find voltage in table

mov si, AGPVoltageGPIOTableLen

;data in al already

call SetOverrideVoltage

clc

SetAGPVoltageExit:

pop di

pop si

pop ax

ret

SetAGPVoltage endp

2.5.5 Chipset Core Voltage

2.5.5.1 GetCSCoreVoltage

Output: data is saved in ECX. Carry clear means success, else failure.

GetCSCoreVoltage proc near

```
push    ax
push    bx
```

```
call    GetVoltMarginOutenStatus
jnc     GetStandardCSCoreVolt
```

```
mov     si, CSCoreVoltageGPIOTableLen
mov     di, (offset CS:CSCoreVoltageGPIOTable - offset CS:SMI_Handler_Start)
;find voltage in table
call    GetOverrideGPIOVoltage
jc      GetStandardCSCoreVolt
```

```
mov     bx, (offset CS:CSCoreVoltageTable - offset CS:SMI_Handler_Start)
;find voltage in table
shl     ax, 2
add     bx, ax
mov     ecx, cs:[bx]          ;Set voltage to ecx
```

```
jmp     CSCoreVoltConvert
GetStandardCSCoreVolt:
mov     ecx, CSCORE_DEFAULT_VOLTAGE_1500
CSCoreVoltConvert:
```

```
clc
pop     bx
pop     ax
ret
```

GetCSCoreVoltage endp

2.5.5.2 SetCSCoreVoltage

Entry: ECX has the value to set. Exit: Carry set means failure.

SetCSCoreVoltage proc near

```
    push    ax
    push    si
    push    di
    ;find voltage in table
    mov     di, (offset CS:CSCoreVoltageTable - offset CS:SMI_Handler_Start)
;find voltage in table
    call    FindVoltageInTable
    jc     SetCSCoreVoltageExit

    call    GetVoltMarginOutenStatus
    jc     SetCSCoreVoltageMarginEnabled

    call    EnableDefaultVoltMargin
SetCSCoreVoltageMarginEnabled:
    mov     di, (offset CS:CSCoreVoltageGPIOTable - offset CS:SMI_Handler_Start)
;find voltage in table
    mov     si, CSCoreVoltageGPIOTableLen
    ;data in al already
    call    SetOverrideVoltage

    cld
SetCSCoreVoltageExit:
    pop     di
    pop     si
    pop     ax
    ret
SetCSCoreVoltage endp
```


2.5.6 CPU Voltage

2.5.6.1 SetCPUVID

Entry: ECX has the value to set. Exit: Carry set means failure.

SetCPUVID proc near

```
push    ax
push    di
push    si
```

```
mov     al, cl; give VID to al
mov     di, (offset CS:CPUVoltageGPIOTable - offset CS:SMI_Handler_Start)
;find voltage in table
mov     si, CPUVoltageGPIOTableLen
```

```
call    SetOverrideVoltage
```

```
call    EnableVIDOverrideCtrl
clc
```

```
pop     si
pop     di
pop     ax
ret
```

SetCPUVID endp

2.5.6.2 GetCPUVID

Output: data is saved in ECX. Carry clear means success, else failure.

GetCPUVID proc near

```
    push    ax
    push    bx

    call    EnableHWMonitor
    jc      GetCPUVIDError
    mov     bl, HW_MONITOR_SMBUS_READ_ADDR
    mov     al, CPU_VID_SMBUS_OFFSET
    call    CS_I2cReadByte
    jc      GetCPUVIDError
    mov     cl, bh
    mov     bl, HW_MONITOR_SMBUS_READ_ADDR
    mov     al, CPU_VID4_SMBUS_OFFSET
    call    CS_I2cReadByte
    jc      GetCPUVIDError
    and     bh, 01h
    shl     bh, 4
    and     cl, 0Fh
    or      bh, cl
    movzx   ax, bh

    movzx   ecx, ax

    clc
GetCPUVIDError:
    pop     bx
    pop     ax

    ret
GetCPUVID endp
```

2.6 Fan Functions

2.6.1 CPU Fan Functions

Note: The Reference board has the CPU fan hardwired to always on full speed. Board modifications are necessary to adjust this fan on the reference board (PCB 180-7R012-0000-D00).

2.6.1.1 GetCPUFanSpeed

Output: data is saved in ECX. Carry clear means success, else failure.

GetCPUFanSpeed proc near

```
push    eax
push    edx
push    bx
```

;Enable the Tachometer register

```
mov     ah, FAN1_TACHO_GP31_CTRL_INDEX
call    SuperIORegRead
or      al, 05h    ;enable the Tachometer register.
mov     ah, FAN1_TACHO_GP31_CTRL_INDEX
call    SuperIORegWrite
```

;Set preload to 0

```
mov     al, 0
mov     ah, FAN1_PRELOAD_REG
call    SuperIORegWrite    ;Set preload
```

mov dl, 03h ;max divisor is 11h

```
mov     ah, FAN_CONTROL    ;Get divisor
```

;Update divisor to max.

```
call    SuperIORegRead
ror     al, 4
and     al, 0FCh
or      al, dl
rol     al, 4
call    SuperIORegWrite
```

```
xor     cx, cx
```

CPUFanIODelay:

```
newiodelay
loop    CPUFanIODelay
```

PROPRIETARY INFORMATION
DO NOT DISTRIBUTE

```

    mov     cx, 10                ;read 10 values
    mov     bl, 0                ;counter for how many FF read
ReadCPUFanTachoAgain:
    newiodelay
    mov     ah, FAN1_TACHO_GP31_REG
    call    SuperIORegRead
    cmp     al, 0FFh

    jne     GotCPUFanTacho
    inc     bl
    loop    ReadCPUFanTachoAgain
GotCPUFanTacho:
    xor     ecx, ecx              ;preset return speed to 0
    cmp     bl, 9
    ja      CPUFanSpeed0
    ;Got Fan tacho reading
    mov     cl, dl                ;give divisor to cl
    mov     ah, 0
    shl     ax, cl                ;multiply by divisor
    shl     ax, 1                ;multiply by 2 because there is 2 pulse per revolution.
    mov     cx, ax                ;put operand in cx
    ;Adjust quantity to be as dx:ax format.
    mov     eax, FAN_TACHO_FREQ
    rol     eax, 16
    mov     dx, ax
    rol     eax, 16

    div     cx
    movzx   ecx, ax                ;quotient is in eax.
CPUFanSpeed0:

    cld
    pop     bx
    pop     edx
    pop     eax
    ret
GetCPUFanSpeed endp

```

2.6.1.2 GetCPUFanPower

Exit: ECX has the CPU Fan power, carry clear means success.

GetCPUFanPower proc near

push ax

;Check To see if Fan1 control is activated.

mov ah, FAN1_GPIO33_CTRL_INDEX

call SuperIORegRead

mov ecx, FAN_FULL_POWER ;preset to full power.

test al, 4 ;Bit 2 enables Fan control

jz GetCPUFanPowerExit

mov ah, FAN1_POWER_INDEX

call SuperIORegRead

test al, 1 ;If Fan1 pin is high

jnz GetCPUFanPowerExit ;yes, then jump with full power return

shl al, 1 ;only bit 1 to 6 are used for fan power.

; and change data range within 00 and FE instead of 0-63.

xor ah, ah

movzx ecx, ax

GetCPUFanPowerExit:

clc

pop ax

ret

2.6.1.3 SetCPUFanPower

Entry: CX (actually CL) has the CPU Fan power to set. Exit: Carry clear means success.

```
SetCPUFanPower proc near
    jmp     DefaultFunc
```

```
ifdef SETCPUFANPOWER_DISABLE
```

```
    push    ax
```

```
    ;Make Fan1 control active.
```

```
    mov     ah, FAN1_GPIO33_CTRL_INDEX
```

```
    mov     al, 04h                ; bit 2 enable fan control
```

```
    call    SuperIORegWrite
```

```
    push    cx
```

```
    shr     cl, 1                ;Bit 1 to 6 is used for fan power
```

```
    and     cl, FAN_POWER_MASK
```

```
    mov     ah, FAN1_POWER_INDEX
```

```
    call    SuperIORegRead
```

```
    and     al, not FAN_POWER_MASK
```

```
    or      al, cl
```

```
    btr     al, 0                ;clear bit 0 for using duty cycle.
```

```
    pop     cx
```

```
    cmp     cl, FAN_FULL_POWER    ;Check if full power required
```

```
    jne     SetCPUFanPowerSIOWrite ; no them jump to write register
```

```
    bts     al, 0                ;yes, then Set bit 0 to 1 for full power.
```

```
SetCPUFanPowerSIOWrite:
```

```
    mov     ah, FAN1_POWER_INDEX
```

```
    call    SuperIORegWrite
```

```
    pop     ax
```

```
    ret
```

```
endif
```

```
SetCPUFanPower endp
```

2.6.2 System Fan Functions

2.6.2.1 GetSecondFanSpeed

Output: data is saved in ECX. Carry clear means success, else failure.

GetSecondFanSpeed proc near

```
push    eax
push    edx
push    bx

;Enable the Tachometer register
mov     ah, FAN2_TACHO_GP30_CTRL_INDEX
call    SuperIORegRead
or      al, 05h    ;enable the Tachometer register.
mov     ah, FAN2_TACHO_GP30_CTRL_INDEX
call    SuperIORegWrite

;Set preload to 0
mov     al, 0
mov     ah, FAN2_PRELOAD_REG
call    SuperIORegWrite    ;Set preload

mov     dl, 03h    ;max divisor is 11h
```

SecondTryAnotherDiv:

```
mov     ah, FAN_CONTROL    ;Get divisor
;Update divisor to max.
call    SuperIORegRead
ror     al, 6
and     al, 0FCh
or      al, dl
rol     al, 6
call    SuperIORegWrite
```

```
xor     cx, cx
```

SecondFanIODelay:

```
newiodelay
loop    SecondFanIODelay
```

```
mov     cx, 10    ;read 10 values
mov     bl, 0     ;counter for how many FF read
```

ReadSecondFanTachoAgain:

**PROPRIETARY INFORMATION
DO NOT DISTRIBUTE**

```

newiodelay
mov     ah, FAN2_TACHO_GP30_REG
call    SuperIORegRead
cmp     al, 0FFh

jne     GotSecondFanTacho
inc     bl
loop    ReadSecondFanTachoAgain

```

GotSecondFanTacho:

```

xor     ecx, ecx           ;preset return speed to 0
cmp     bl, 9
jna     ComputeSecondFanRPM
dec     dl
cmp     dl, 0FFh
je      SecondFanSpeed0
jmp     SecondTryAnotherDiv

```

ComputeSecondFanRPM:

```

;Got Fan tachometer reading
mov     cl, dl             ;give divisor to cl
mov     ah, 0
shl     ax, cl             ;multiply by divisor
shl     ax, 1             ;multiply by 2 because there is 2 pulse per revolution.
mov     cx, ax             ;put operand in cx
;Adjust quantity to be as dx:ax format.
mov     eax, FAN_TACHO_FREQ
rol     eax, 16
mov     dx, ax
rol     eax, 16

div     cx
movzx   ecx, eax           ;quotient is in eax.

```

SecondFanSpeed0:

```

clc
pop     bx
pop     edx
pop     eax

```

ret

GetSecondFanSpeed endp

2.6.2.2 GetSecondFanPower

Exit: ECX has the CPU Fan power, carry clear means success.

GetSecondFanPower proc near

```
    push    ax
    ;Check To see if Fan1 control is activated.
    mov     ah, FAN2_GPIO32_CTRL_INDEX
    call    SuperIORegRead
    mov     ecx, FAN_FULL_POWER      ;preset to full power
    test    al, 4                    ;Bit 2 enables fan control
    jz      GetSecondFanPowerExit

    mov     ah, FAN2_POWER_INDEX
    call    SuperIORegRead
    test    al, 1                    ;If Fan1 pin is high
    jnz     GetSecondFanPowerExit   ;yes, then jump with full power return
    shl     al, 1                    ;only bit 1 to 6 are used for fan power.
    ; and change data range within 00 and FE instead of 0-63.
    xor     ah, ah
```

```
    movzx   ecx, ax
```

GetSecondFanPowerExit:

```
    clc
    pop     ax
    ret
```

GetSecondFanPower endp

**PROPRIETARY INFORMATION
DO NOT DISTRIBUTE**

2.6.2.3 SetSecondFanPower

Entry: ECX (bit 0 to 5) has the CPU Fan power to set. Exit: Carry clear means success

SetSecondFanPower proc near

```
    push    ax
```

```
    ;Make Fan2 control active.
```

```
    mov     ah, FAN2_GPIO32_CTRL_INDEX
```

```
    mov     al, C17_FAN2_CTRL_ENABLE
```

```
    ; bit 2 enable fan control bit 0 invert output
```

```
    call    IsC18DBoard
```

```
    jnc     NoInvertOutput
```

```
    mov     al, C18D_FAN2_CTRL_ENABLE
```

```
NoInvertOutput:
```

```
    call    SuperIORegWrite
```

```
    push    cx
```

```
    shr     cl, 1                                ;Bit 1 to 6 is used for fan power
```

```
    and     cl, FAN_POWER_MASK
```

```
    mov     ah, FAN2_POWER_INDEX
```

```
    call    SuperIORegRead
```

```
    and     al, not FAN_POWER_MASK
```

```
    or      al, cl
```

```
    btr     al, 0                                ;clear bit 0 for using duty cycle.
```

```
    pop     cx
```

```
    cmp     cl, FAN_FULL_POWER                ;Check if full power required
```

```
    jne     SetSecondFanPowerSIOWrite        ;no then jump to write register
```

```
    bts     al, 0                                ;yes, then Set bit 0 to 1 for full power.
```

```
SetSecondFanPowerSIOWrite:
```

```
    mov     ah, FAN2_POWER_INDEX
```

```
    call    SuperIORegWrite
```

```
    pop     ax
```

```
    ret
```

SetSecondFanPower endp

2.7 CMOS Functions

The following are used to support the “Access BIOS” page functions.

2.7.1 WPTComputeCMOSChecksumAll

Calculate CMOS checksum from 10h to 2Eh and 40h to 7Bh. Exit: Clear carry.

WPTComputeCMOSChecksumAll proc near

```
    push    cx
    push    dx

    mov     cl, 10h NMI_OFF
    mov     dl, 2Eh NMI_OFF
    call    WPTComputeCMOSChecksum

    mov     cl, 40h NMI_OFF
    mov     dl, 7Bh NMI_OFF
    call    WPTComputeCMOSChecksum

    clc
    pop     dx
    pop     cx
    ret
```

WPTComputeCMOSChecksumAll endp

2.7.2 WPTComputeCMOSChecksum

Calculate CMOS checksum from CL to DL. ENTRY: CL: start offset, DL: end offset

WPTComputeCMOSChecksum proc near

```

push    bx
        xor     bx, bx
WPTCMOS_Checksum:                          ;Cmos Check Sum [40h-7Ah]
        mov     al, cl
        call    SMI_Get_Cmos
        mov     ah, 0
        add     bx, ax
        inc     cl
        cmp     cl, dl
        je      SaveChecksum
        jmp     WPTCMOS_Checksum
SaveChecksum:
        mov     al, dl
        mov     ah, bh
        call    SMI_Set_Cmos

        mov     al, dl
        inc     al
        mov     ah, bl
        call    SMI_Set_Cmos

        pop     bx
        ret
WPTComputeCMOSChecksum endp

```

2.7.3 Motherboard Information

2.7.3.1 ReadMBInfo

Read motherboard vendor, motherboard model and version

; Entry: None

; Exit: Data will be saved in scratch register 1-7.

; scratch 1: bit0-15: vendor ID

; bit 16-31: version

; scratch 2-7: will be used for motherboard model, one

; one letter with ASCII encoding.

; Modified:

;

```
ReadMBInfo    proc near
    push      dx
    push      eax

    mov       dx, SMI_IO_BASE_ADDR+SW_SMI_SCRATCH1_REG
    call      GetBoardVersion
    shl       eax, 16
    mov       ax, BOARD_VENDOR_ID
    mov       ecx, eax           ;N09

    mov       dx, SMI_IO_BASE_ADDR+SW_SMI_SCRATCH2_REG
    mov       eax, BOARD_MODEL1
    out       dx, eax

    mov       dx, SMI_IO_BASE_ADDR+SW_SMI_SCRATCH3_REG
    mov       eax, BOARD_MODEL2
    out       dx, eax

    pop       eax
    pop       dx
    ret
ReadMBInfo    endp
```

**PROPRIETARY INFORMATION
DO NOT DISTRIBUTE**

2.7.3.2 GetBoardVersion

Get board version

```
; Entry: None
; Exit: version is saved in ax
; Modified:
;
GetBoardVersion proc near
    mov     ax, BOARD_VERSION
    ret
GetBoardVersion endp
```

2.7.3.3 Read BIOS Information

```
;-----
;
; ReadBIOSInfo: Read BIOS vendor, Version, and date
;
; Entry: None
; Exit: Data will be saved in scratch register 1-7.
;   scratch 1: bit0-15: vendor ID same as Motherboard Vendor ID.
;           bit 16-31: version
;   scratch 2 and 3: BIOS version represented as ASCIIcode
;   scratch 4: BIOS date represented as number.
;           Bit 31:16 : Year;
;           Bit 15:8  : Month
;           Bit 7:0   : Date
; Modified:
; Note: If this function is not supported, nTune will not display
;       corresponding information.
;
ReadBIOSInfo  proc near
    push  dx
    push  eax

    mov   ecx, BIOS_VENDOR_ID           ;Scratch register 1 is passed through ecx

    mov   dx, SMI_IO_BASE_ADDR+SW_SMI_SCRATCH2_REG
    mov   eax, BIOS_VERSION1
    out   dx, eax

    mov   dx, SMI_IO_BASE_ADDR+SW_SMI_SCRATCH3_REG
    xor   eax, eax
    ;mov   eax, BIOS_VERSION2
```

```

out    dx, eax

mov     dx, SMI_IO_BASE_ADDR+SW_SMI_SCRATCH4_REG
mov     eax, (BIOS_YEAR shl 16) + (BIOS_MONTH shl 8) + BIOS_DATE
out     dx, eax

pop     eax
pop     dx

ret
ReadBIOSInfo endp

```

2.8 Source Definitions

```

WPT_INTERFACE_VERSION      EQU    0100h ;Version for WPT interface

CPU_TEMP_THRESHOLD        EQU    90 ; CPU warning threshold.
MB_TEMP_THRESHOLD         EQU    43 ; MB warning threshold.

#ifdef K8_CPU_SUPPORT
SMBUS_PORT2                EQU    SMBus2_Port
#else
SMBUS_PORT2                EQU    SMBus_Port_2
#endif

SUPERIO_GPIO_PORT_BASE    EQU    Superio_GPIO_Port ;IO mapped
superIO base

HW_MONITOR_SMBUS_READ_ADDR EQU    5Bh ;SMBUS address for read hardware
monitor data
HW_MONITOR_SMBUS_WRITE_ADDR EQU    5Ah ;SMBUS address for read hardware
monitor data

SMSC_LPC47M15_SMB_REG_CONFIGURATION EQU    40h ;HW monitor config
register.
CPU_TEMP_SMBUS_OFFSET      EQU    26h ;SMBUS offset to CPU Temp
BOARD_TEMP_SMBUS_OFFSET    EQU    27h ;SMBUS offset to Motherboard
Temp
VOLTMARGINEN_GPIO43_CTRL   EQU    3Eh ;Voltage Margin out enable (GPIO
43) control register.
VOLTMARGINEN_GPIO_GROUP    EQU    4Eh ;Voltage Margin out enable (GPIO
43) Data register
VOLTMARGINEN_GPIOMASK      EQU    08h
MARGINOUTEN_ENABLE_STATUS  EQU    08h
MARGINOUTEN_ENABLE_BIT     EQU    3 ;Bit 3

```

**PROPRIETARY INFORMATION
DO NOT DISTRIBUTE**

GPIO_CTRL_OUTPUT	EQU	00h	;Default GPIO control register value
GPIO_VID_CTRL_OUTPUT for CPU VID control	EQU	80h	;Default GPIO control register value
GPIO_CTRL_OUTPUT_MASK	EQU	05h	
AUX_MARGIN1_GPIO12_CTRL	EQU	25h	
AUX_MARGIN2_GPIO17_CTRL	EQU	2Ah	
AUX_GPIO_GROUP	EQU	4Bh	
AUX_GPIOMASK	EQU	84h	
AUX_DEFAULT_VOLTAGE_1500	EQU	3FC00000h	;Default is 1.5v
AUX_MARGIN1_DEFAULT_STATUS	EQU	00h	
AUX_MARGIN1_BIT	EQU	2	;Bit 2
AUX_MARGIN2_DEFAULT_STATUS	EQU	00h	
AUX_MARGIN2_BIT	EQU	7	;Bit 7
MEM_MARGIN1_GPIO15_CTRL	EQU	28h	
MEM_MARGIN2_GPIO16_CTRL	EQU	29h	
MEM_GPIO_GROUP	EQU	4Bh	
MEM_GPIOMASK	EQU	60h	
MEM_DEFAULT_VOLTAGE_2600	EQU	40266666h	;Default is 2.6v
MEM_MARGIN1_DEFAULT_STATUS	EQU	20h	
MEM_MARGIN1_BIT	EQU	5	;Bit 5
MEM_MARGIN2_DEFAULT_STATUS	EQU	40h	
MEM_MARGIN2_BIT	EQU	6	;Bit 6
AGP_MARGIN1_GPIO13_CTRL	EQU	26h	
AGP_MARGIN2_GPIO14_CTRL	EQU	27h	
AGP_GPIO_GROUP	EQU	4Bh	
AGP_GPIOMASK	EQU	18h	
AGP_DEFAULT_VOLTAGE_1500	EQU	3FC00000h	;Default is 1.5v
AGP_MARGIN1_DEFAULT_STATUS	EQU	08h	
AGP_MARGIN1_BIT	EQU	3	;Bit 3
AGP_MARGIN2_DEFAULT_STATUS	EQU	10h	
AGP_MARGIN2_BIT	EQU	4	;Bit 4
CSCORE_MARGIN1_GPIO10_CTRL	EQU	23h	
CSCORE_MARGIN2_GPIO11_CTRL	EQU	24h	
CSCORE_GPIO_GROUP	EQU	4Bh	
CSCORE_GPIOMASK	EQU	03h	
CSCORE_DEFAULT_VOLTAGE_1500	EQU	3FC00000h	;Default is 1.5v
CSCORE_MARGIN1_DEFAULT_STATUS	EQU	01h	
CSCORE_MARGIN1_BIT	EQU	0	;Bit 0
CSCORE_MARGIN2_DEFAULT_STATUS	EQU	02h	
CSCORE_MARGIN2_BIT	EQU	1	;Bit 1



**PROPIETARY INFORMATION
DO NOT DISTRIBUTE**

VID_OVERRIDE_GPIO55_CTRL control register.	EQU	44h ;VID override enable (GPIO 55)
VID_OVERRIDE_GPIO_GROUP Data reg.	EQU	4Fh ;VID override enable (GPIO 55)
VID_OVERRIDE_GPIOMASK	EQU	20h ;bit 5 is for GPIO 55
CPUVID_OVERRIDE0_GPIO25_CTRL	EQU	30h
CPUVID_OVERRIDE1_GPIO26_CTRL	EQU	31h
CPUVID_OVERRIDE2_GPIO20_CTRL	EQU	2Bh
CPUVID_OVERRIDE3_GPIO21_CTRL	EQU	2Ch
CPUVID_OVERRIDE4_GPIO22_CTRL	EQU	2Dh
CPUVID_GPIO_GROUP	EQU	4Ch
CPUVID_GPIOMASK	EQU	67h
CPU_VID_SMBUS_OFFSET bit 0-3	EQU	47h ;SMBUS Offset to read VID
CPU_VID4_SMBUS_OFFSET bit 4	EQU	49h ;SMBUS offset to read VID
CPUVID0_GPIO_BIT	EQU	5 ;gpio 25
CPUVID1_GPIO_BIT	EQU	6
CPUVID2_GPIO_BIT	EQU	0
CPUVID3_GPIO_BIT	EQU	1
CPUVID4_GPIO_BIT	EQU	2
CPUVID_DEFAULT_STATUS	EQU	0
FAN1_GPIO33_CTRL_INDEX	EQU	36h
FAN2_GPIO32_CTRL_INDEX	EQU	35h
C17_FAN2_CTRL_ENABLE update in code.	EQU	04h ;C17 board uses 04h ;dynamically
C18D_FAN2_CTRL_ENABLE ;dynamically update in code.	EQU	06h ;C18D board uses 06h
FAN1_POWER_INDEX	EQU	56h
FAN2_POWER_INDEX	EQU	57h
FAN_FULL_POWER	EQU	0FFh
FAN_POWER_MASK	EQU	7Eh
FAN1_TACHO_GP31_CTRL_INDEX	EQU	34h
FAN2_TACHO_GP30_CTRL_INDEX	EQU	33h
FAN_CONTROL	EQU	58h
FAN1_TACHO_GP31_REG	EQU	59h
FAN2_TACHO_GP30_REG	EQU	5Ah
FAN1_PRELOAD_REG	EQU	5Bh
FAN2_PRELOAD_REG	EQU	5Ch
FAN_TACHO_FREQ	EQU	1DFFB0h
;SMI related define		
SMI_IO_BASE_ADDR	EQU	SYSTEM_PORT

PROPRIETARY INFORMATION
DO NOT DISTRIBUTE

```

SW_SMI_STATUS_REG          EQU    2Fh
SW_SMI_SCRATCH0_REG        EQU    80h
SW_SMI_SCRATCH1_REG        EQU    84h
SW_SMI_SCRATCH2_REG        EQU    88h
SW_SMI_SCRATCH3_REG        EQU    8Ch
SW_SMI_SCRATCH4_REG        EQU    90h
SW_SMI_SCRATCH5_REG        EQU    94h
SW_SMI_SCRATCH6_REG        EQU    98h
SW_SMI_SCRATCH7_REG        EQU    9Ch

; Function return code
WPT_RETURN_INVALID         EQU    0FFh
WPT_RETURN_FAIL            EQU    0FEh
WPT_RETURN_SUCCESS         EQU    000h

BOARD_VENDOR_ID             EQU    01h    ; NVidia
BOARD_VERSION               EQU    200h    ; 2.0

#ifdef K8_CPU_SUPPORT
BOARD_MODEL1                EQU    "NVK8"
BOARD_MODEL2                EQU    "-CRB"
else
BOARD_MODEL1                EQU    "NVK7"
BOARD_MODEL2                EQU    "-CRB"
#endif ;K8_CPU_SUPPORT

;Software SMI command. Used in MCP_PM_SWISMI_REGISTER
NV_WPT                      EQU    0C8h

;Voltage table end indicator
VOLTAGE_TABLE_END          EQU    80000000h

```

PROPRIETARY INFORMATION
DO NOT DISTRIBUTE

CONFIDENTIAL INFORMATION
DO NOT COPY

CONFIDENTIAL INFORMATION
DO NOT COPY

Information furnished is believed to be accurate and reliable. However, NVIDIA Corporation assumes no responsibility for the consequences of use of such information or for any infringement of patents or other rights of third parties that may result from its use. No license is granted by implication or otherwise under any patent or patent rights of NVIDIA Corporation. Specifications mentioned in this publication are subject to change without notice. This publication supersedes and replaces all information previously supplied. NVIDIA Corporation products are not authorized for use as critical components in life support devices or systems without express written approval of NVIDIA Corporation.

Trademarks

NVIDIA and the NVIDIA logo are registered trademarks and nForce is a trademark of NVIDIA Corporation. Microsoft, Windows and the Windows logo are registered trademarks of Microsoft Corporation. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

Copyright NVIDIA Corporation 2008