Advanced Configuration and Power Interface Specification Revision 2.0 Errata

Compaq Computer Corporation
Intel Corporation
Microsoft Corporation
Phoenix Technologies Ltd.
Toshiba Corporation

Errata document revision 1.0
September 19, 2000

Revision	Change Description	Affected Sections
1.0	Initial errata document for ACPI 2.0.	
	Re-inserted mistakenly deleted sentence fragment.	5.2
	FADT SCI_INT field - clarified to be the SCI interrupts's Global System Interrupt number when no 8259 exists in the system.	5.2.8
	Incorrect reference to Processor declaration section.	5.2.10.5
	Local APIC Address Override Structure length field corrected.	5.2.10.11
	I/O SAPIC Strucure - length field corrected, Global System Interrupt Base and I/O SAPIC Address field descriptions expanded/clarified.	5.2.10.12
	Local SAPIC Structure flags length corrected to 4 from 2. Other offsets adjusted accordingly. Incorrect reference to Processor declaration section.	5.2.10.13
	_CS4 critical thermal trip point renamed to _HOT	12.4, 12.5
	Corrected Embedded Controller method name - removed trailing numbers	14.2
	LNOT(Logical Not) evaluation result correction.	16.2.3.4.2.26
	ASL macro for fixed I/O port descriptor listed incorrectly in previous section.	16.2.4.5, 16.2.4.6
	AML Root-Path only encoding for NamePath was missing as was NullName	17.2.1

5.2 Device Power State Definitions

Device power states are states of particular devices; as such, they are generally not visible to the user. For example, some devices may be in the Off state even though the system as a whole is in the Working state.

Device states apply to any device on any bus. They are generally defined in terms of four principal criteria:

- **Power consumption.** How much power the device uses.
- **Device context.** How much of the context of the device is retained by the hardware. The OS is responsible for restoring any lost device context (this may be done by resetting the device).
- **Device driver.** What the device driver must do to restore the device to full on.
- **Restore time.** How long it takes to restore the device to full on.

The device power states are defined below, although very generically. Many devices do not have all four power states defined. Devices may be capable of several different low-power modes, but if there is no user-perceptible difference between the modes only the lowest power mode will be used. The Device Class Power Management Specifications, included in Appendix A of this specification, describe which of these power states are defined for a given type (class) of device and define the specific details of each power state for that device class. For a list of the available *Device Class Power Management Specifications*, see "Appendix A: Device Class Specifications."

D3 Off

Power has been fully removed from the device. The device context is lost when this state is entered, so the OS software will reinitialize the device when powering it back on. Since device context and power are lost, devices in this state do not decode their address lines. Devices in this state have the longest restore times. All classes of devices define this state.

D2

The meaning of the D2 Device State is defined by each device class. Many device classes may not define D2. In general, D2 is expected to save more power and preserve less device context than D1 or

D0. Buses in D2 may cause the device to lose some context (for example, by reducing power on the bus, thus forcing the device to turn off some of its functions).

D1

The meaning of the D1 Device State is defined by each device class. Many device classes may not define D1. In general, D1 is expected to save less power and preserve more device context than D2.

D0 Fully-On

This state is assumed to be the highest level of power consumption. The device is completely active and responsive, and is expected to remember all relevant context continuously.

Table 2-2 Summary of Device Power States

Device State	Power Consumption	Device Context Retained	Driver Restoration
D0 - Fully-On	As needed for operation	All	None
D1	D0>D1>D2>D3	>D2	<D2
D2	D0>D1>D2>D3	<D1	>D1
D3 - Off	0	None	Full initialization and load

Note: Devices often have different power modes within a given state. Devices can use these modes as long as they can automatically transparently switch between these modes from the software, without violating the rules for the current D*x* state the device is in. Low-power modes that adversely affect performance (in other words, low speed modes) or that are not transparent to software cannot be done automatically in hardware; the device driver must issue commands to use these modes.

5.2.8 Fixed ACPI Description Table (FADT)

The Fixed ACPI Description Table (FADT) defines various fixed hardware ACPI information vital to an ACPI-compatible OS, such as the base address for the following hardware registers blocks: PM1a_EVT_BLK, PM1b_EVT_BLK, PM1a_CNT_BLK, PM1b_CNT_BLK, PM2_CNT_BLK, PM_TMR_BLK, GPE0_BLK, and GPE1_BLK.

The FADT also has a pointer to the DSDT that contains the Differentiated Definition Block, which in turn provides variable information to an ACPI-compatible OS concerning the base system design.

All fields in the FADT that provide hardware addresses provide processor-relative physical addresses.

Table 5-8 Fixed ACPI Description Table (FADT) Format

Field	Byte Length	Byte Offset	Description
Header			
Signature	4	0	'FACP'. Signature for the Fixed ACPI Description Table.
Length	4	4	Length, in bytes, of the entire FADT.
Revision	1	8	3
Checksum	1	9	Entire table must sum to zero.
OEMID	6	10	OEM ID
OEM Table ID	8	16	For the FADT, the table ID is the manufacture model ID. This field must match the OEM Table ID in the RSDT.
OEM Revision	4	24	OEM revision of FADT for supplied OEM Table ID.
Creator ID	4	28	Vendor ID of utility that created the table. For tables containing Definition Blocks, this is the ID for the ASL Compiler.
Creator Revision	4	32	Revision of utility that created the table. For tables containing Definition Blocks, this is the revision for the ASL Compiler.
FIRMWARE_CTRL	4	36	Physical memory address (0-4 GB) of the FACS, where OSPM and Firmware exchange control information. See section 5.2.6, "Root System Description Table," for a description of the FACS.
DSDT	4	40	Physical memory address (0-4 GB) of the DSDT.
Reserved	1	44	ACPI 1.0 defined this offset as a field named INT_MODEL, which has been eliminated in ACPI 2.0.as operating systems to date have had no use for this field. New systems should set this field to zero but field values of one are also allowed to maintain compatibility with ACPI 1.0.

Table 5-8 Fixed ACPI Description Table (FADT) Format *(continued)*

Field	Byte Length	Byte Offset	Description
Preferred_PM_Profile	1	45	This field is set by the OEM to convey the preferred power management profile to OSPM. OSPM can use this field to set default power management policy parameters during OS installation. Field Values: 0–Unspecified 1–Desktop 2–Mobile 3–Workstation 4–Enterprise Server 5–SOHO Server 6–Appliance PC >6–Reserved
SCI_INT	2	46	System vector the SCI interrupt is wired to in 8259 mode. On systems that do not contain the 8259, this field contains the Global System interrupt number of the SCI interrupt. OSPM is required to treat the ACPI SCI interrupt as a sharable, level, active low interrupt.
SMI_CMD	4	48	System port address of the SMI Command Port. During ACPI OS initialization, OSPM can determine that the ACPI hardware registers are owned by SMI (by way of the SCI_EN bit), in which case the ACPI OS issues the ACPI_ENABLE command to the SMI_CMD port. The SCI_EN bit effectively tracks the ownership of the ACPI hardware registers. OSPM issues commands to the SMI_CMD port synchronously from the boot processor. This field is reserved and must be zero on system that does not support System Management mode.
ACPI_ENABLE	1	52	The value to write to SMI_CMD to disable SMI ownership of the ACPI hardware registers. The last action SMI does to relinquish ownership is to set the SCI_EN bit. During the OS initialization process, OSPM will synchronously wait for the transfer of SMI ownership to complete, so the ACPI system releases SMI ownership as quickly as possible. This field is reserved and must be zero on systems that do not support Legacy Mode.

Table 5-8 Fixed ACPI Description Table (FADT) Format *(continued)*

Field	Byte Length	Byte Offset	Description
ACPI_DISABLE	1	53	The value to write to SMI_CMD to re-enable SMI ownership of the ACPI hardware registers. This can only be done when ownership was originally acquired from SMI by OSPM using ACPI_ENABLE. An OS can hand ownership back to SMI by relinquishing use to the ACPI hardware registers, masking off all SCI interrupts, clearing the SCI_EN bit and then writing ACPI_DISABLE to the SMI_CMD port from the boot processor. This field is reserved and must be zero on systems that do not support Legacy Mode.
S4BIOS_REQ	1	54	The value to write to SMI_CMD to enter the S4BIOS state. The S4BIOS state provides an alternate way to enter the S4 state where the firmware saves and restores the memory context. A value of zero in S4BIOS_F indicates S4BIOS_REQ is not supported. (See Table 5-12.)
PSTATE_CNT	1	55	The value OSPM writes to the SMI_CMD register to assume processor performance state control responsibility.
PM1a_EVT_BLK	4	56	System port address of the PM1a Event Register Block. See section 4.7.3.1, “PM1 Event Grouping,” for a hardware description layout of this register block. This is a required field. This field is superseded in ACPI 2.0 by the X_PM1a_EVT_BLK field.
PM1b_EVT_BLK	4	60	System port address of the PM1b Event Register Block. See section 4.7.3.1, “PM1 Event Grouping,” for a hardware description layout of this register block. This field is optional; if this register block is not supported, this field contains zero. This field is superseded in ACPI 2.0 by the X_PM1b_EVT_BLK field.
PM1a_CNT_BLK	4	64	System port address of the PM1a Control Register Block. See section 4.7.3.2, “PM1 Control Grouping,” for a hardware description layout of this register block. This is a required field. This field is superseded in ACPI 2.0 by the X_PM1a_CNT_BLK field.
PM1b_CNT_BLK	4	68	System port address of the PM1b Control Register Block. See section 4.7.3.2, “PM1 Control Grouping,” for a hardware description layout of this register block. This field is optional; if this register block is not supported, this field contains zero. This field is superseded in ACPI 2.0 by the X_PM1b_CNT_BLK field.
PM2_CNT_BLK	4	72	System port address of the PM2 Control Register Block. See section 4.7.3.4, “PM2 Control (PM2_CNT),” for a hardware description layout of this register block. This field is optional; if this register block is not supported, this field contains zero. This field is superseded in ACPI 2.0 by the X_PM2_CNT_BLK field.

Table 5-8 Fixed ACPI Description Table (FADT) Format *(continued)*

Field	Byte Length	Byte Offset	Description
PM_TMR_BLK	4	76	System port address of the Power Management Timer Control Register Block. See section 4.7.3.3, "Power Management Timer (PM_TMR)," for a hardware description layout of this register block. This is a required field. This field is superseded in ACPI 2.0 by the X_PM_TMR_BLK field.
GPE0_BLK	4	80	System port address of General-Purpose Event 0 Register Block. See section 5.2.8, "Fixed ACPI Description Table," for a hardware description of this register block. This is an optional field; if this register block is not supported, this field contains zero. This field is superseded in ACPI 2.0 by the X_GPE0_BLK field.
GPE1_BLK	4	84	System port address of General-Purpose Event 1 Register Block. See section 5.2.8, "Fixed ACPI Description Table," for a hardware description of this register block. This is an optional field; if this register block is not supported, this field contains zero. This field is superseded in ACPI 2.0 by the X_GPE1_BLK field.
PM1_EVT_LEN	1	88	Number of bytes decoded by PM1a_EVT_BLK and, if supported, PM1b_ EVT_BLK. This value is ≥ 4.
PM1_CNT_LEN	1	89	Number of bytes decoded by PM1a_CNT_BLK and, if supported, PM1b_CNT_BLK. This value is ≥ 1.
PM2_CNT_LEN	1	90	Number of bytes decoded by PM2_CNT_BLK. Support for the PM2 register block is optional. If supported, this value is ≥ 1. If not supported, this field contains zero.
PM_TMR_LEN	1	91	Number of bytes decoded by PM_TMR_BLK. This field's value must be 4.
GPE0_BLK_LEN	1	92	Number of bytes decoded by GPE0_BLK. The value is a non-negative multiple of 2.
GPE1_BLK_LEN	1	93	Number of bytes decoded by GPE1_BLK. The value is a non-negative multiple of 2.
GPE1_BASE	1	94	Offset within the ACPI general-purpose event model where GPE1 based events start.
CST_CNT	1	95	The value OSPM writes to the SMI_CMD register to indicate OS support for the _CST object and C States Changed notification.
P_LVL2_LAT	2	96	The worst-case hardware latency, in microseconds, to enter and exit a C2 state. A value > 100 indicates the system does not support a C2 state.
P_LVL3_LAT	2	98	The worst-case hardware latency, in microseconds, to enter and exit a C3 state. A value > 1000 indicates the system does not support a C3 state.

Table 5-8 Fixed ACPI Description Table (FADT) Format *(continued)*

Field	Byte Length	Byte Offset	Description
FLUSH_SIZE	2	100	If WBINVD=0, the value of this field is the number of flush strides that need to be read (using cacheable addresses) to completely flush dirty lines from any processor's memory caches. Notice that the value in FLUSH_STRIDE is typically the smallest cache line width on any of the processor's caches (for more information, see the FLUSH_STRIDE field definition). If the system does not support a method for flushing the processor's caches, then FLUSH_SIZE and WBINVD are set to zero. Notice that this method of flushing the processor caches has limitations, and WBINVD=1 is the preferred way to flush the processors caches. This value is typically at least 2 times the cache size. The maximum allowed value for FLUSH_SIZE multiplied by FLUSH_STRIDE is 2 MB for a typical maximum supported cache size of 1 MB. Larger cache sizes are supported using WBINVD=1. This value is ignored if WBINVD=1. This field is maintained for ACPI 1.0 processor compatibility on existing systems. Processors in new ACPI 2.0-compatible systems are required to support the WBINVD function and indicate this to OSPM by setting the WBINVD field = 1.
FLUSH_STRIDE	2	102	If WBINVD=0, the value of this field is the cache line width, in bytes, of the processor's memory caches. This value is typically the smallest cache line width on any of the processor's caches. For more information, see the description of the FLUSH_SIZE field. This value is ignored if WBINVD=1. This field is maintained for ACPI 1.0 processor compatibility on existing systems. Processors in new ACPI 2.0-compatible systems are required to support the WBINVD function and indicate this to OSPM by setting the WBINVD field = 1.
DUTY_OFFSET	1	104	The zero-based index of where the processor's duty cycle setting is within the processor's P_CNT register.

Table 5-8 Fixed ACPI Description Table (FADT) Format *(continued)*

Field	Byte Length	Byte Offset	Description
DUTY_WIDTH	1	105	The bit width of the processor's duty cycle setting value in the P_CNT register. Each processor's duty cycle setting allows the software to select a nominal processor frequency below its absolute frequency as defined by: THTL_EN = 1 BF * DC/(2^{DUTY_WIDTH}) Where: BF–Base frequency DC–Duty cycle setting When THTL_EN is 0, the processor runs at its absolute BF. A DUTY_WIDTH value of 0 indicates that processor duty cycle is not supported and the processor continuously runs at its base frequency.
DAY_ALRM	1	106	The RTC CMOS RAM index to the day-of-month alarm value. If this field contains a zero, then the RTC day of the month alarm feature is not supported. If this field has a non-zero value, then this field contains an index into RTC RAM space that OSPM can use to program the day of the month alarm. See section 4.7.2.4, "Real Time Clock Alarm," for a description of how the hardware works.
MON_ALRM	1	107	The RTC CMOS RAM index to the month of year alarm value. If this field contains a zero, then the RTC month of the year alarm feature is not supported. If this field has a non-zero value, then this field contains an index into RTC RAM space that OSPM can use to program the month of the year alarm. If this feature is supported, then the DAY_ALRM feature must be supported also.
CENTURY	1	108	The RTC CMOS RAM index to the century of data value (hundred and thousand year decimals). If this field contains a zero, then the RTC centenary feature is not supported. If this field has a non-zero value, then this field contains an index into RTC RAM space that OSPM can use to program the centenary field.
IAPC_BOOT_ARCH	2	109	IA-PC Boot Architecture Flags. See Table 5-10 for a description of this field.
Reserved	1	111	Must be 0.
Flags	4	112	Fixed feature flags. See Table 5-9 for a description of this field.

Table 5-8 Fixed ACPI Description Table (FADT) Format *(continued)*

Field	Byte Length	Byte Offset	Description
RESET_REG	12	116	The address of the reset register represented in Generic Address Structure format (See section 4.7.3.6, "Reset Register," for a description of the reset mechanism.) **Note**: Only System I/O space, System Memory space and PCI Configuration space (bus #0) are valid for values for Address_Space_ID. Also, Register_Bit_Width must be 8 and Register_Bit_Offset must be 0.
RESET_VALUE	1	128	Indicates the value to write to the RESET_REG port to reset the system. (See section 4.7.3.6, "Reset Register," for a description of the reset mechanism.)
Reserved	3	129	Must be 0.
X_FIRMWARE_CTRL	8	132	64bit physical address of the FACS.
X_DSDT	8	140	64bit physical address of the DSDT.
X_PM1a_EVT_BLK	12	148	Extended address of the PM1a Event Register Block, represented in Generic Address Structure format. See section 4.7.3.1, "PM1 Event Grouping," for a hardware description layout of this register block. This is a required field.
X_PM1b_EVT_BLK	12	160	Extended address of the PM1b Event Register Block, represented in Generic Address Structure format. See section 4.7.3.1, "PM1 Event Grouping," for a hardware description layout of this register block. This field is optional; if this register block is not supported, this field contains zero.
X_PM1a_CNT_BLK	12	172	Extended address of the PM1a Control Register Block, represented in Generic Address Structure format. See section 4.7.3.2, "PM1 Control Grouping," for a hardware description layout of this register block. This is a required field.
X_PM1b_CNT_BLK	12	184	Extended address of the PM1b Control Register Block, represented in Generic Address Structure format. See section 4.7.3.2, "PM1 Control Grouping," for a hardware description layout of this register block. This field is optional; if this register block is not supported, this field contains zero.
X_PM2_CNT_BLK	12	196	Extended address of the Power Management 2 Control Register Block, represented in Generic Address Structure format. See section 4.7.3.4, "PM2 Control (PM2_CNT)," for a hardware description layout of this register block. This field is optional; if this register block is not supported, this field contains zero.

Table 5-8 Fixed ACPI Description Table (FADT) Format *(continued)*

Field	Byte Length	Byte Offset	Description
X_PM_TMR_BLK	12	208	Extended address of the Power Management Timer Control Register Block, represented in Generic Address Structure format. See section 4.7.3.3, "Power Management Timer (PM_TMR)," for a hardware description layout of this register block. This is a required field.
X_GPE0_BLK	12	220	Extended address of the General-Purpose Event 0 Register Block, represented in Generic Address Structure format. See section 5.2.8, "Fixed ACPI Description Table," for a hardware description of this register block. This is an optional field; if this register block is not supported, this field contains zero.
X_GPE1_BLK	12	232	Extended address of the General-Purpose Event 1 Register Block, represented in Generic Address Structure format. See section 5.2.8, "Fixed ACPI Description Table," for a hardware description of this register block. This is an optional field; if this register block is not supported, this field contains zero.

5.2.10.5 Processor Local APIC

When using the APIC interrupt model, each processor in the system is required to have a Processor Local APIC record and an ACPI Processor object. OSPM does not expect the information provided in this table to be updated if the processor information changes during the lifespan of an OS boot. While in the sleeping state, processors are not allowed to be added, removed, nor can their APIC ID or Flags change. When a processor is not present, the Processor Local APIC information is either not reported or flagged as disabled.

Table 5-17 Processor Local APIC Structure

Field	Byte Length	Byte Offset	Description
Type	1	0	0–Processor Local APIC structure
Length	1	1	8
ACPI Processor ID	1	2	The ProcessorId for which this processor is listed in the ACPI Processor declaration operator. For a definition of the Processor operator, see section 16.2.3.3.1.17~~6~~, "Processor (Declare Processor)."
APIC ID	1	3	The processor's local APIC ID.
Flags	4	4	Local APIC flags. See Table 5-18 for a description of this field.

5.2.10.11 Local APIC Address Override Structure

This optional structure overrides the physical address of the local APIC in the MADT'stable header using the Generic Address Structure.

If defined, OSPM must use the address specified in this structure for all local APICs (and local SAPICs), rather than the address contained in the MADT's table header. Only one Local APIC Address Override Structure may be defined.

Table 5-24 Local APIC Address Override Structure

Field	Byte Length	Byte Offset	Description
Type	1	0	5–Local APIC Address Override Structure
Length	1	1	12~~6~~
Reserved	2	2	Reserved (must be set to zero)
Local APIC Address	8	4	Physical address of Local APIC

5.2.10.12 I/O SAPIC Structure

The I/O SAPIC structure is very similar to the I/O APIC structure. If both I/O APIC and I/O SAPIC structures exist for a specific APIC ID, the information in the I/O SAPIC structure must be used.

The I/O SAPIC structure uses the I/O_APIC_ID field as defined in the I/O APIC table. The Vector_Base field remains unchanged but has been moved. The I/O APIC address has been deleted. A new address and reserved field have been added.

Table 5-25 I/O SAPIC Structure

Field	Byte Length	Byte Offset	Description
Type	1	0	6–I/O SAPIC Structure
Length	1	1	16~~20~~
I/O APIC ID	1	2	I/O SAPIC ID
Reserved	1	3	Reserved (must be zero)
Global System Interrupt Base	4	4	The global system interrupt number where this I/O APIC's interrupt inputs start. The number of interrupt inputs is determined by the I/O APIC's *Max Redir Entry* register.~~Global System Interrupt Base~~
I/O SAPIC Address	8	8	The 64-bit physical address to access this I/O SAPIC. Each I/O SAPIC resides at a unique address.~~Physical address for I/O SAPIC~~

If defined, OSPM must use the information contained in the I/O SAPIC structure instead of the information from the I/O APIC structure.

If both I/O APIC and an I/O SAPIC structures exist in an MADT, the OEM/BIOS writer must prevent "mixing" I/O APIC and I/O SAPIC addresses. This is done by ensuring that there are at least as many I/O SAPIC structures as I/O APIC structures and that every I/O APIC structure has a corresponding I/O SAPIC structure (same APIC ID).

5.2.10.13 Local SAPIC Structure

The Processor local SAPIC structure is very similar to the processor local APIC structure. When using the SAPIC interrupt model, each processor in the system is required to have a Processor Local SAPIC record and an ACPI Processor object. OSPM does not expect the information provided in this table to be updated if the processor information changes during the lifespan of an OS boot. While in the sleeping state, processors are not allowed to be added, removed, nor can their SAPIC ID or Flags change. When a processor is not present, the Processor Local SAPIC information is either not reported or flagged as disabled.

Table 5-26 Processor Local SAPIC Structure

Field	Byte Length	Byte Offset	Description
Type	1	0	7–Processor Local SAPIC structure
Length	1	1	8
ACPI Processor ID	2	2	The Processor Id listed in the processor object. For a definition of the Processor object, see section 16.2.3.3.1.1~~6~~7, "Processor (Declare Processor)."
Flags	~~2~~4	4	Local SAPIC flags. See Table 5-18 for a description of this field.
Local SAPIC ID	1	~~6~~8	The processor's local SAPIC ID
Local SAPIC EID	1	~~7~~9	The processor's local SAPIC EID

12.4 Thermal Zone Object Requirements

While not all thermal zone objects are required to be present in each thermal zone defined in the namespace, OSPM levies conditional requirements for the presence of specific thermal zone objects based on the definition of other related thermal zone objects. These requirements are outlined below:

- All thermal zones must contain the _TMP object.
- A thermal zone must define at least one trip point: _CRT, _HOT~~CS4~~, _ACx, or _PSV.
- If _ACx is defined then an associated _ALx must be defined (e.g. defining _AC0 requires _AL0 also be defined).
- If _PSV is defined then either _PSL or _TZD must be defined. _PSL and _TZD may both be defined.
- If _PSL is defined then:

If a performance control register is defined (via either P_BLK or _PTC) for a processor defined in _PSL then _TC1, _TC2, and _TSP must be defined.
If a performance control register is not defined (via either P_BLK or _PTC) for a processor defined in _PSL then the processor must support processor performance states (in other words, the processor's processor object must include _PCT, _PSS, and _PPC).

- If _PSV is defined and _PSL is not defined (in other words, only _TZD is defined) then at least one device in the _TZD device list must support device performance states.
- _SCP is optional.
- _TZD is optional outside of the _PSV requirement outlined above.
- If _HOT~~CS4~~ is defined then the system must support the S4 sleeping state.

12.5 Thermal Zone Examples

12.5.1 Example: The Basic Thermal Zone

The following ASL describes a basic configuration where the entire system is treated as a single thermal zone. Cooling devices for this thermal zone consist of a processor and one single-speed fan. This is an example only.

Notice that this thermal zone object (TZ0) is defined in the _SB scope. Thermal zone objects should appear in the namespace under the portion of the system that comprises the thermal zone. For example, a thermal zone that is isolated to a docking station should be defined within the scope of the docking station device. Besides providing for a well-organized namespace, this configuration allows OSPM to dynamically adjust its thermal policy as devices are added or removed from the system.

```
Scope(\_SB) {
    Processor(
        CPU0,
        1,          // unique number for this processor
        0x110,      // system IO address of Pblk Registers
        0x06        // length in bytes of PBlk
        ) {}
```

```
Scope(\_SB.PCI0.ISA0) {
    Device(EC0) {
        Name(_HID, EISAID("PNP0C09"))    // ID for this EC
        // current resource description for this EC
        Name(_CRS,
                    ResourceTemplate() {
                        IO(Decode16,0x62,0x62,0,1)
                        IO(Decode16,0x66,0x66,0,1)
            })
        Name(_GPE, 0)      // GPE index for this EC

        // create EC's region and field for thermal support
        OperationRegion(EC0, EmbeddedControl, 0, 0xFF)
        Field(EC0, ByteAcc, Lock, Preserve) {
            MODE,   1,      // thermal policy (quiet/perform)
            FAN,    1,      // fan power (on/off)
            ,       6,      // reserved
            TMP,    8,      // current temp
            AC0,    8,      // active cooling temp (fan high)
            ,       8,      // reserved
            PSV,    8,      // passive cooling temp
            HOTCS4      8,      // critical S4 temp
            CRT,    8       // critical temp
            }

        // following is a method that OSPM will schedule after
        // it receives an SCI and queries the EC to receive value 7
        Method(_Q07) {
            Notify (\_SB.PCI0.ISA0.EC0.TZ0, 0x80)
        }   // end of Notify method

        // fan cooling on/off - engaged at AC0 temp
        PowerResource(PFAN, 0, 0) {
            Method(_STA) { Return (\_SB.PCI0.ISA0.EC0.FAN) }     // check power state
            Method(_ON) { Store (One, \_SB.PCI0.ISA0.EC0.FAN) }         // turn on fan
            Method(_OFF) { Store ( Zero, \_SB.PCI0.ISA0.EC0.FAN) }      // turn off fan
        }

        // Create FAN device object
        Device (FAN) {
            // Device ID for the FAN
            Name(_HID, EISAID("PNP0C0B"))
            // list power resource for the fan
            Name(_PR0, Package(){PFAN})
        }

        // create a thermal zone
        ThermalZone (TZ0) {
            Method(_TMP) { Return (\_SB.PCI0.ISA0.EC0.TMP )} // get current temp
            Method(_AC0) { Return (\_SB.PCI0.ISA0.EC0.AC0) } // fan high temp
            Name(_AL0, Package(){\_SB.PCI0.ISA0.EC0.FAN})    // fan is act cool dev
            Method(_PSV) { Return (\_SB.PCI0.ISA0.EC0.PSV) } // passive cooling temp
            Name(_PSL, Package (){\_SB.CPU0})                // passive cooling devices
            Method(_HOTCS4) { Return (\_SB.PCI0.ISA0.EC0.HOTCS4) }      // get critical S4
    temp
            Method(_CRT) { Return (\_SB.PCI0.ISA0.EC0.CRT) } // get critical temp
            Method(_SCP, 1) { Store (Arg1, \_SB.PCI0.ISA0.EC0.MODE) } // set cooling mode
            Name(_TC1, 4)                                    // bogus example constant
            Name(_TC2, 3)                                    // bogus example constant
            Name(_TSP, 150)                                  // passive sampling = 15 sec
            Name(_TZP, 0)                                    // polling not required
        } // end of TZ0

    }   // end of ECO
}   // end of \_SB.PCI0.ISA0 scope-

} // end of \_SB scope
```

12.5.2 Example: Multiple-Speed Fans

The following ASL describes a thermal zone consisting of a processor and one dual-speed fan. As with the previous example, this thermal zone object (TZ0) is defined in the _SB scope and represents the entire system. This is an example only.

```
Scope(\_SB) {
    Processor(
        CPU0,
        1,          // unique number for this processor
        0x110,      // system IO address of Pblk Registers
        0x06        // length in bytes of PBlk
        ) {}


Scope(\_SB.PCI0.ISA0) {
    Device(EC0) {
        Name(_HID, EISAID("PNP0C09"))    // ID for this EC
        // current resource description for this EC
        Name(_CRS,
                    ResourceTemplate() {
                        IO(Decode16,0x62,0x62,0,1)
                        IO(Decode16,0x66,0x66,0,1)
            })
        Name(_GPE, 0)      // GPE index for this EC

        // create EC's region and field for thermal support
        OperationRegion(EC0, EmbeddedControl, 0, 0xFF)
        Field(EC0, ByteAcc, Lock, Preserve) {
            MODE,   1,      // thermal policy (quiet/perform)
            FAN0,   1,      // fan strength high/off
            FAN1,   1,      // fan strength low/off
            ,       5,      // reserved
            TMP,    8,      // current temp
            AC0,    8,      // active cooling temp (high)
            AC1,    8,      // active cooling temp (low)
            PSV,    8,      // passive cooling temp
            HOTCS4      8,      // critical S4 temp
            CRT,    8       // critical temp
        }

        // following is a method that OSPM will schedule after it
        // receives an SCI and queries the EC to receive value 7
        Method(_Q07) {
            Notify (\_SB.PCI0.ISA0.EC0.TZ0, 0x80)
        } end of Notify method

        // fan cooling mode high/off - engaged at AC0 temp
        PowerResource(FN10, 0, 0) {
            Method(_STA) { Return (\_SB.PCI0.ISA0.EC0.FAN0) }    // check power state
            Method(_ON) { Store (One, \_SB.PCI0.ISA0.EC0.FAN0) } // turn on fan at high
            Method(_OFF) { Store (Zero, \_SB.PCI0.ISA0.EC0.FAN0) }// turn off fan
        }

        // fan cooling mode low/off   - engaged at AC1 temp
        PowerResource(FN11, 0, 0) {
            Method(_STA) { Return (\_SB.PCI0.ISA0.EC0.FAN1) }    // check power state
            Method(_ON) { Store (One, \_SB.PCI0.ISA0.EC0.FAN1) } // turn on fan at low
            Method(_OFF) { Store (Zero, \_SB.PCI0.ISA0.EC0.FAN1) }// turn off fan
        }
```

```
        // Following is a single fan with two speeds.  This is represented
        // by creating two logical fan devices.  When FN2 is turned on then
        // the fan is at a low speed.  When FN1 and FN2 are both on then
        // the fan is at high speed.
        //
        // Create FAN device object FN1
        Device (FN1) {
            // Device ID for the FAN
            Name(_HID, EISAID("PNP0C0B"))
            Name(_PR0, Package(){FN10, FN11})
        }

        // Create FAN device object FN2
        Device (FN2) {
            // Device ID for the FAN
            Name(_HID, EISAID("PNP0C0B"))
            Name(_PR0, Package(){FN10})
        }

        // create a thermal zone
        ThermalZone (TZ0) {
            Method(_TMP) { Return (\_SB.PCI0.ISA0.EC0.TMP )}  // get current temp
            Method(_AC0) { Return (\_SB.PCI0.ISA0.EC0.AC0) }  // fan high temp
            Method(_AC1) { Return (\_SB.PCI0.ISA0.EC0.AC1) }  // fan low temp
            Name(_AL0, Package() {\_SB.PCI0.ISA0.EC0.FN1})    // active cooling (high)
            Name(_AL1, Package() {\_SB.PCI0.ISA0.EC0.FN2})    // active cooling (low)
            Method(_PSV) { Return (\_SB.PCI0.ISA0.EC0.PSV) }  // passive cooling temp
            Name(_PSL, Package() {\_SB.CPU0})                 // passive cooling devices
            Method(_HOTCS4) { Return (\_SB.PCI0.ISA0.EC0.HOTCS4) } // get critical S4 temp
            Method(_CRT) { Return (\_SB.PCI0.ISA0.EC0.CRT) } // get crit. temp
            Method(_SCP, 1) { Store (Arg1, \_SB.PCI0.ISA0.EC0.MODE) } // set cooling mode
            Name(_TC1, 4)                                     // bogus example constant
            Name(_TC2, 3)                                     // bogus example constant
            Name(_TSP, 150)                                   // passive sampling = 15 sec
            Name(_TZP, 0)                                     // polling not required
        } // end of TZ0

    } // end of ECO
}   // end of \_SB.PCI0.ISA0 scope


} // end of \_SB scope
```

14.2 Declaring SMBus Host Controller Objects

EC-based SMBus 1.0-compatible HCs should be modeled in the ACPI namespace as described in section 13.12, “Defining an Embedded Controller SMBus Host Controller in ACPI Namespace.” An example definition is given below. Using the HID value “ACPI0001” identifies that this SMB-HC is implemented on an embedded controller using the standard SMBus register set defined in section 13.9, SMBus Host Controller Interface via Embedded Controller.”

```
Device (SMB0)
{
    Name(_HID, "ACPI0001")        // EC-based SMBus 1.0 compatible Host Controller
    Name(_EC0, 0x2030)            // EC offset 0x20, query bit 0x30
      :
}
```

EC-based SMBus 2.0-compatible host controllers should be defined similarly in the name space as follows:

```
Device (SMB0)
{
    Name(_HID, "ACPI0005")        // EC-based SMBus 2.0 compatible Host Controller
    Name(_EC1, 0x2030)            // EC offset 0x20, query bit 0x30
      :
}
```

Non–EC-based SMB-HCs should be modeled in a manner similar to the EC-based SMBus HC. An example definition is given below. These devices use a vendor-specific hardware identifier (HID) to specify the type of SMB-HC (do not use “ACPI0001” or “ACPI0005”). Using a vendor-specific HID allows the correct software to be loaded to service this segment’s SMBus address space.

```
Device(SMB0)
{
    Name(_HID, "<Vendor-Specific HID>")   // Vendor-Specific HID
      :
}
```

Regardless of the type of hardware, some OS software element (for example, the SMBus HC driver) must register with OSPM to support all SMBus operation regions defined for the segment. This software allows the generic SMBus interface defined in this section to be used on a specific hardware implementation by translating between the conceptual (for example, SMBus address space) and physical (for example, process of writing/reading registers) models. Because of this linkage, SMBus operation regions must be defined immediately within the scope of the corresponding SMBus device.

16.2.3.4.2.26 LNot (Logical Not)

```
LNotTerm                    := LNot(
                                  Source,              //TermArg=>Integer
                               ) => Boolean
```

Source1 is evaluated as an integer. If the value is ~~non-~~zero True is returned; otherwise, False is returned.

16.2.4.5 ASL Macro for I/O Port Descriptor

The following macro generates a short I/O descriptor:

```
IO(
    Decode16 | Decode10,          // _DEC
    WordConstExpr,                // _MIN, Address minimum
    WordConstExpr,                // _MAX, Address max
    ByteConstExpr,                // _ALN, Base alignment
    ByteConstExpr                 // _LEN, Range length
    NameString | Nothing          // A name to refer back to this resource

)
```

~~The following macro generates a short Fixed I/O descriptor:~~

```
~~FixedIO(~~
~~    WordConstExpr,              // _BAS, Address base~~
~~    ByteConstExpr               // _LEN, Range length~~
~~    NameString | Nothing        // A name to refer back to this resource~~
~~    )~~
```

16.2.4.6 ASL Macro for Fixed I/O Port Descriptor

The following macro generates a short Fixed I/O descriptor:

```
FixedIO(
    WordConstExpr,              // _BAS, Address base
    ByteConstExpr               // _LEN, Range length
    NameString | Nothing        // A name to refer back to this resource
    )
```

16.2.4.7 ASL Macro for Short Vendor-Defined Descriptor

The following macro generates a short Vendor-Defined descriptor:

```
VendorShort(
    NameString | Nothing          // A name to refer back to this resource
    )
    {
    ByteConstExpr [, ByteConstExpr ...]   // List of bytes, up to 7 bytes
        }
```

17.2.1 Name Objects Encoding

```
LeadNameChar                := 'A'-'Z' | '_'
DigitChar                   := '0'-'9'
NameChar                    := DigitChar | LeadNameChar
RootChar                    := '\'
ParentPrefixChar            := '^'

'A'-'Z'                     := 0x41-0x5a
'_'                         := 0x5f
'0'-'9'                     := 0x30-0x39
'\'                         := 0x5c
'^'                         := 0x5e

NameSeg                     := <LeadNameChar NameChar NameChar NameChar>
                               // Notice that NameSegs shorter than 4 characters are
                               // filled with trailing '_'s.
NameString                  := <RootChar NamePath> | <PrefixPath NamePath>
PrefixPath                  := Nothing | <'^' PrefixPath>
NamePath                    := NameSeg | DualNamePath | MultiNamePath | NullName

DualNamePath                := DualNamePrefix NameSeg NameSeg
DualNamePrefix              := 0x2e
MultiNamePath               := MultiNamePrefix SegCount NameSeg(SegCount)
MultiNamePrefix             := 0x2f
SegCount                    := ByteData
                               // SegCount can be from 1 to 255.
                               // MultiNamePrefix(35) => 0x2f 0x23
                               // and following by 35 NameSegs.
                               // So, the total encoding length
                               // will be 1 + 1 + 35*4 = 142.
                               // Notice that:
                               //   DualNamePrefix NameSeg NameSeg
                               // has a smaller encoding than the
                               // equivalent encoding of:
                               //    MultiNamePrefix(2) NameSeg NameSeg
SimpleName                  := NameString | ArgObj | LocalObj
SuperName                   := SimpleName | DebugObj | Type6Opcode
NullName                    := 0x00
```

```
Target                          := SuperName | NullName
```