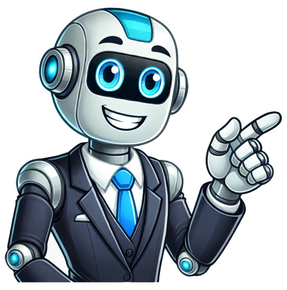
Continue



ACPI Specification This chapter describes the ACPI thermal model and specifies the ACPI Namespace objects OSPM uses for thermal management of the platform. ACPI defines interfaces that allow OSPM to be proactive in its system cooling policies. With OSPM in control of the operating environment, cooling decisions can be made based on the system's application load, the user's preference towards performance or energy conservation, and thermal heuristics. Graceful shutdown of devices or the entire system at critical heat levels becomes possible as well. The following sections describe the ACPI thermal model and the ACPI Namespace objects available to OSPM to apply platform thermal management policy. The ACPI thermal model is based around conceptual platform regions called thermal zones that physically contain devices, thermal sensors, and cooling controls. Generally speaking, the entire platform is one large thermal zone, but the platform can be partitioned into several ACPI thermal zones if necessary to enable optimal thermal management. ACPI Thermal zones are a logical collection of interfaces to temperature sensors, trip points, thermal property information, and thermal controls. Thermal zone interfaces apply either thermal zone wide or to specific devices, including processors, contained within the thermal zone. ACPI defines namespace objects that provide the thermal zone-wide interfaces in Section 11.4. A subset of these objects may also be defined under devices. OS implementations compatible with the ACPI 3.0 thermal model, interface with these objects but also support OS native device driver interfaces that perform similar functions at the device level. This allows the integration of devices with embedded thermal sensors and controls, perhaps not accessible by AML, to participate in the ACPI thermal model through their inclusion in the ACPI thermal zone. OSPM is responsible for applying an appropriate thermal policy when a thermal zone contains both thermal objects and native OS device driver interfaces for thermal control. Some devices in a thermal zone may be comparatively large producers of thermal load in relation to other devices in the thermal zone. Devices may also have varying degrees of thermal sensitivity. For example, some devices may tolerate operation at a significantly higher temperature than other devices. As such, the platform can provide OSPM with information about the platform's device topology and the resulting influence of one device's thermal load generation on another device. This information must be comprehended by OSPM for it to achieve optimal thermal management through the application of cooling controls. ACPI expects all temperatures to be represented in tenths of degrees. This resolution is deemed sufficient to enable OSPM to perform robust platform thermal management. Fig. 11.1 ACPI Thermal Zone¶ There are three cooling policies that OSPM uses to control the thermal state of the hardware. The policies are active, passive and critical. Active Cooling. OSPM takes a direct action such as turning on one or more fans. Applying active cooling controls typically consume power and produce some amount of noise, but are able to cool a thermal zone without limiting system performance. Active cooling temperature trip points declare the temperature thresholds OSPM uses to decide when to start or stop different active cooling devices. Passive Cooling. OSPM reduces the power consumption of devices to reduce the temperature of a thermal zone, such as slowing (throttling) the processor clock. Applying passive cooling controls typically produces no user-noticeable noise. Passive cooling temperature trip points specify the temperature thresholds where OSPM will start or stop passive cooling. Critical Trip Points. These are threshold temperatures at which OSPM performs an orderly, but critical, shutdown of a device or the entire system. The _HOT object declares the critical temperature at which OSPM may choose to transition the system into the S4 sleeping state, if supported, The _CRT object declares the critical temperature at which OSPM must perform a critical shutdown. When a thermal zone appears in the ACPI Namespace or when a new device becomes a member of a thermal zone, OSPM retrieves the temperature thresholds (trip points) at which it executes a cooling policy. When OSPM receives a temperature change notification, it evaluates the thermal zone's temperature interfaces to retrieve current temperature values. OSPM compares the current temperature values against the temperature thresholds. If any temperature is greater than or equal to a corresponding active trip point then OSPM will perform active cooling . If any temperature is greater than or equal to a corresponding passive trip point then OSPM will perform passive cooling. If the _TMP object returns a value greater than or equal to the value returned by the _HOT object then OSPM may choose to transition the system into the S4 sleeping state, if supported. If the _TMP object returns a value greater than or equal to the value returned by the _CRT object then OSPM must shut the system down. Embedded Hot and Critical trip points may also be exposed by individual devices within a thermal zone. Upon passing of these trip points, OSPM must decide whether to shut down the device or the entire system based upon device criticality to system operation. OSPM must also evaluate the thermal zone's temperature interfaces when any thermal zone appears in the namespace (for example, during system initialization) and must initiate a cooling policy as warranted independent of receipt of a temperature change notification. This allows OSPM to cool systems containing a thermal zone whose temperature has already exceeded temperature thresholds at initialization time. An optimally designed system that uses several thresholds can notify OSPM of thermal increase or decrease by raising an event every several degrees. This enables OSPM to anticipate thermal trends and incorporate heuristics to better manage the system's temperature. To implement a preference towards performance or energy conservation, OSPM can request that the platform change the priority of active cooling (performance) versus passive cooling (energy conservation/silence) by evaluating the _SCP (Set Cooling Policy) object for the thermal zone or a corresponding OS-specific interface to individual devices within a thermal zone. The platform or its devices can change the active and passive cooling temperature trip points and notify OSPM to reevaluate the trip point interfaces to establish the new policy threshold settings. The following are the primary uses for this type of thermal notification: When OSPM changes the platform's cooling policy from one cooling mode to another. When a swappable bay device is inserted or removed. A swappable bay is a slot that can accommodate several different devices that have identical form factors, such as a CD-ROM drive, disk drive, and so on. Many mobile PCs have this concept already in place. After the crossing of an active or passive trip point is signaled to implement hysteresis. In each situation, OSPM must be notified to re-evaluate the thermal zone's trip points via the AML code execution of a Notify(thermal_zone, 0x81) statement or via an OS specific interface invoked by device drivers for zone devices participating in the thermal model. When OSPM changes the platform's cooling policy from one cooling mode to the other, the following occurs: OSPM notifies the platform of the new cooling mode by running the Set Cooling Policy (_SCP) control method in all thermal zones and invoking the OS-specific Set Cooling Policy interface to all participating devices in each thermal zone. Thresholds are updated in the hardware and OSPM is notified of the change. OSPM re-evaluates the active and passive cooling temperature trip points for the zone and all devices in the zone to obtain the new temperature thresholds. The platform can adjust the thermal zone temperature to accommodate the maximum operating temperature of a bay device as necessary. For example: Hardware detects that a device was inserted into or removed from the bay, updates the temperature thresholds, and then notifies OSPM of the thermal policy change and device insertion events. OSPM re-enumerates the devices and re-evaluates the active and passive cooling temperature trip points. An OEM can build hysteresis into platform thermal design by dynamically resetting cooling temperature thresholds. For example: When the temperature increases to the designated threshold, OSPM will turn on the associated active cooling device or perform passive cooling. The platform resets the threshold value to a lower temperature (to implement hysteresis) and notifies OSPM of the change. Because of this new threshold value, the fan will be turned off at a lower temperature than when it was turned on (therefore implementing a negative hysteresis). When the temperature hits the lower threshold value, OSPM will turn off the associated active cooling device or cease passive cooling. The hardware will reset _ACx to its original value and notify OSPM that the trip points have once again been altered. The ability of the platform and its devices to asynchronously notify an ACPI-compatible OS of meaningful changes in the thermal zone's temperature is a highly desirable capability that relieves OSPM from implementing a poll-based policy and generally results in a much more responsive and optimal thermal policy implementation. Each notification instructs OSPM to evaluate whether a trip point has been crossed and allows OSPM to anticipate temperature trends for the thermal zone. It is recognized that much of the hardware used to implement thermal zone functionality today is not capable of generating ACPI-visible notifications (SCIs) or only can do so with wide granularity (for example, only when the temperature crosses the critical threshold). In these environments, OSPM must poll the thermal zone's temperature periodically to implement an effective policy. While ACPI specifies a mechanism that enables OSPM to poll thermal zone temperature, platform reliance on thermal zone polling is strongly discouraged by this specification. OEMs should design systems that asynchronously notify OSPM whenever a meaningful change in the zone's temperature occurs - relieving OSPM of the overhead associated with polling. In some cases, embedded controller firmware can overcome limitations of existing thermal sensor capabilities to provide the desired asynchronous notification. Notice that the _TZP (thermal zone polling) object is used to indicate whether a thermal zone must be polled by OSPM, and if so, a recommended polling frequency. See _TZP (Thermal Zone Polling) for more information. Thermal zone-wide temperature sensor hardware that supports asynchronous temperature change notifications does so using an SCI. The AML code that responds to this SCI must execute a Notify(thermal_zone, 0x80) statement to inform OSPM that a meaningful change in temperature has occurred. Alternatively, devices with embedded temperature sensors may signal their associated device drivers and the drivers may use an OS-specific interface to signal OSPM's thermal policy driver. A device driver may also invoke a device specific control method that executes a Notify(thermal_zone, 0x80) statement. When OSPM receives this thermal notification, it will evaluate the thermal zone's temperature interfaces to evaluate the current temperature values. OSPM will then compare the values to the corresponding cooling policy trip point values (either zone-wide or device-specific). If the temperature has crossed over any of the policy thresholds, then OSPM will actively or passively cool (or stop cooling) the system, or shut the system down entirely. Both the number and granularity of thermal zone trip points are OEM-specific. However, it is important to notice that since OSPM can use heuristic knowledge to help cool the system, the more events OSPM receives the better understanding it will have of the system's thermal characteristic. Fig. 11.2 Thermal Events¶ For example, the simple thermal zone illustrated above includes hardware that will generate a temperature change notification using a 5° Celsius granularity. All thresholds (_PSV, _AC1, _AC0, and _CRT) exist within the monitored range and fall on 5 boundaries. This granularity is appropriate for this system as it provides sufficient opportunity for OSPM to detect when a threshold is crossed as well as to understand the thermal zone's basic characteristics (temperature trends). Note: The ACPI specification defines Kelvin as the standard unit for absolute temperature values. All thermal zone objects must report temperatures in Kelvin when reporting absolute temperature values. All figures and examples in this section of the specification use Celsius for reasons of clarity. ACPI allows Kelvin to be declared in precision of 1/10th of a degree (for example, 310.5). Kelvin is expressed as follows: \[\theta /K = T/(degrees Celsius) + 273.2\] Temperature sensor hardware that is incapable of generating thermal change events, or that can do so for only a few thresholds should inform OSPM to implement a poll-based policy. OSPM does this to ensure that temperature changes across threshold boundaries are always detectable. Polling can be done in conjunction with hardware notifications. For example, thermal zone hardware that only supports a single threshold might be configured to use this threshold as the critical temperature trip point. Assuming that hardware monitors the temperature at a finer granularity than OSPM would, this environment has the benefit of being more responsive when the system is overheating. A thermal zone advertises the need to be polled by OSPM via the _TZP object. See _TZP (Thermal Zone Polling) for more information. Active cooling devices typically consume power and produce some amount of noise when enabled. These devices attempt to cool a thermal zone through the removal of heat rather than limiting the performance of a device to address an adverse thermal condition. The active cooling interfaces in conjunction with the active cooling lists or the active cooling relationship table (_ART) allow the platform to use an active device that offers varying degrees of cooling capability or multiple cooling devices. The active cooling temperature trip points designate the temperature where Active cooling is engaged or disengaged (depending upon the direction in which the temperature is changing). For thermal zone-wide active cooling controls, the _ALx object evaluates to a list of devices that actively cool the zone or the _ART object evaluates to describe the entire active cooling relationship of various devices. For example: If a standard single-speed fan is the Active cooling device, then _AC0 evaluates to the temperature where active cooling is engaged and the fan is listed in _AL0. If the zone uses two independently controlled single-speed fans to regulate the temperature, then _AC0 will evaluate to the maximum cooling temperature using two fans, and _AC1 will evaluate to the standard cooling temperature using one fan. If a zone has a single fan with a low speed and a high speed, the _AC0 will evaluate to the temperature associated with running the fan at high-speed, and _AC1 will evaluate to the temperature associated with running the fan at low speed. _AL0 and _AL1 will both point to different device objects associated with the same physical fan, but control the fan at different speeds. If the zone uses two independently controlled multiple-speed fans to regulate the temperature, _AC0 of the target devices evaluates to the temperature at which OSPM will engage fan devices described by the _ART object as needed up to a maximum capability level. For ASL coding examples that illustrate these points, see Thermal Zone Interface Requirements and Thermal Zone Examples. Passive cooling controls are able to cool a thermal zone without creating noise and without consuming additional power (actually saving power), but do so by decreasing the performance of the devices in the zone . The processor passive cooling threshold (_PSV) in conjunction with the processor list (_PSL) allows the platform to indicate the temperature at which a passive control, for example clock throttling, will be applied to the processor(s) residing in a given thermal zone. Unlike other cooling policies, during passive cooling of processors OSPM may take the initiative to actively monitor the temperature in order to cool the platform. On an ACPI-compatible platform that properly implements CPU throttling, the temperature transitions will be similar to the following figure, in a coolable environment, running a coolable workload: Fig. 11.3 Temperature and CPU Performance Versus Time¶ The following equation should be used by OSPM to assess the optimum CPU performance change necessary to lower the thermal zone's temperature: Equation #1 \[\Delta P [\%] = _TC1 * (T_{n} - T_{n-1}) + _TC2 * (T_{n} - T_{t})\] Where: Tt = target temperature (_PSV) The two coefficients _TC1 and _TC2 and the sampling period _TSP are hardware-dependent constants the OEM must supply to OSPM (for more information, see Section 11.4). The _TSP object contains a time interval that OSPM uses to poll the hardware to sample the temperature. Whenever the time value returned by _TSP has elapsed, OSPM will evaluate _TMP to sample the current temperature (shown as Tn in the above equation). Then OSPM will use the sampled temperature and the passive cooling temperature trip point (_PSV) (which is the target temperature Tt) to evaluate the equation for \(\Delta P\). The granularity of \(\Delta P\) is determined by the CPU duty width of the system. Note: Equation #1 has an implied formula. Equation #2: \(P_{n} = P_{n-1} + HW[- ?P]\) \(Minimum \% = 2 are available } } } // end of CPU0 scope Device(CPU1) { Name(_HID, "ACPI0007") Name(_UID, 1) // // Load additional objects if 3.0 Thermal model support is available // Method(_INI, 0) { If (_OSI("3.0 Thermal Model")) { LoadTable("OEM1", "PmRef", "Cpu1", "_SB.CPU1") // 3.0 Thermal Model } } // For brevity, most processor objects have been excluded // from this example (such as _PSS, _CST, _PCT, _PPC, _PTC, etc.) // Processor Throttle Control object Name(_PTC, ResourceTemplate() { Register(SystemIO, 32, 0, 0x120) // Processor Control Register(SystemIO, 32, 0, 0x120) // Processor Status }) // Throttling Supported States // The values shown are for exemplary purposes only Name(_TSS, Package() { // Read: freq percentage, power, latency, control, status Package() {0x64, 1000, 0x0, 0x7, 0x0}, // Throttle off (100%) Package() {0x58, 800, 0x0, 0xF, 0x0}, // 87.5% Package() {0x4B, 600, 0x0, 0xE, 0x0}, // 75% Package() {0x3F, 400, 0x0, 0xD, 0x0} // 62.5% }) // Throttling Present Capabilities // The values shown are for exemplary purposes only Method(_TPC) { If(_SB.AC) {Return(0) // All throttle states available } Else { Return(2) // Throttle states >= 2 are available } } } // end of CPU1 scope Scope(_SB.PCI0.ISA0) { Device(EC0) { Name(_HID, EISAID("PNP0C09")) // ID for this EC // // Load additional objects if 3.0 Thermal model support is available // Method(_INI, 0) { If (_OSI("3.0 Thermal Model")) { LoadTable("OEM1", "PmRef", "Tz3", "_SB.PCI0.ISA0.EC0") // 3.0 Tz } } // 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 , 6, // reserved TMP, 16, // current temp AC0, 16, // active cooling temp PSV, 16, // passive cooling temp HOT, 16, // critical S4 temp CRT, 16 // critical temp } // Following is a method that OSPM will schedule after it // 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 } // Following is a single fan with one speed. // Create FAN device object FN1 Device (FN1) { // Device ID for the FAN Name(_HID, EISAID("PNP0C0B")) Name(_UID, 0) Name(_PR0, Package(){FN10}) } // Receives an SCI and queries the EC to receive value 7 Method(_Q07) { Notify (_SB.PCI0.ISA0.EC0.TZ0, 0x80) } // end of Notify method // Create standard specific thermal zone ThermalZone (TZ0) { Method(_TMP) { Return (_SB.PCI0.ISA0.EC0.TMP)} // get current temp Name(_PSL, Package() {_SB.CPU0, _SB.CPU1}) // passive cooling devices Name(_AL0, Package() {_SB.PCI0.ISA0.EC0.FN1}) // active cooling Method(_AC0) { Return (_SB.PCI0.ISA0.EC0.AC0) } // fan temp (high) Method(_AC1) { Return (_SB.PCI0.ISA0.EC0.AC1) } // fan temp (low) Method(_PSV) { Return (_SB.PCI0.ISA0.EC0.PSV) } // passive cooling temp Method(_HOT) { Return (_SB.PCI0.ISA0.EC0.HOT) } // get critical S4 temp Method(_CRT) { Return (_SB.PCI0.ISA0.EC0.CRT) } // get crit. temp Name(_TC1, 4) // bogus example constant Name(_TC2, 3) // bogus example constant Method(_SCP, 1) { Store (Arg0, _SB.PCI0.ISA0.EC0.MODE) } // set cooling mode Name(_TSP, 150) // passive sampling = 15 sec } // end of TZ0 } // end of ECO } // end of _SB.PCI0.ISA0 scope } // end of _SB scope // // ACPI 3.0 Thermal Model SSDT // DefinitionBlock ("TZASSDT.aml", "OEM1", 0x01, "PmRef", "Tz3", 0x3000) { External(_SB.PCI0.ISA0.EC0, DeviceObj) External(_SB.CPU0, DeviceObj) External(_SB.CPU1, DeviceObj) Scope(_SB.PCI0.ISA0.EC0) { // Create an ACPI 3.0 specific thermal zone ThermalZone (TZ0) { // This TRT is for exemplary purposes only Name(_TRT, Package() { // Thermal relationship package data. A package is generated for // each permutation of device sets. 2 devices = 4 entries. // Read: source, target, thermal influence, sampling period, 4 reserved Package () {_SB.CPU0, _SB.CPU0, 20, 1, 0, 0, 0, 0}, Package () {_SB.CPU0, _SB.CPU1, 10, 15, 0, 0, 0, 0}, Package () {_SB.CPU1, _SB.CPU0, 10, 15, 0, 0, 0, 0}, Package () {_SB.CPU1, _SB.CPU1, 20, 1, 0, 0, 0, 0} }) // end of TRT } // end of TZ0 } // end of EC0 Scope } // end of SSDT // // CPU0 3.0 Thermal Model SSDT // DefinitionBlock ("CPU0SSDT.aml", "OEM1", 0x01, "PmRef", "CPU0", 0x3000) { External(_SB.CPU0, DeviceObj) External(_SB.PCI0.ISA0.TZ0) External(_SB.PCI0.ISA0.TZ0, ThermalZoneObj) Scope(_SB.CPU0) { // // Add the objects required for 3.0 extended thermal support // // Create a region and fields for thermal support; the platform // fills in the values and traps on writes to enable hysteresis. // The Operation Region location is invalid OperationRegion(CP00, SystemMemory, 0x00000000, 0xA) Field(CP00, ByteAcc, Lock, Preserve) { SCP, 1, // thermal policy (passive/active) RTV, 1, // absolute or relative temperature , 6, // reserved AC0, 16, // active cooling temp PSV, 16, // passive cooling temp CRT, 16, // critical temp TPT, 16, // Temp trip point crossed TST, 8 // Temp sensor threshold } Method(_TZM, 0) { Return(_SB.PCI0.ISA0.TZ0) } // thermal zone member // Some thermal zone methods are now located under the // thermal device participating in the 3.0 thermal model. // These methods provide device specific thermal information Method(_SCP, 1) { Store (Arg0, _SB.CPU0.SCP) } // set cooling mode Method(_RTV) { Return (_SB.CPU0.RTV) } // absolute or relative temp Method(_AC0) { Return (_SB.CPU0.AC0) } // active cooling (fan) temp Method(_PSV) { Return (_SB.CPU0.PSV) } // passive cooling temp Method(_CRT) { Return (_SB.CPU0.CRT) } // critical temp Name(_TC1, 4) // thermal constant 1 (INVALID) Name(_TC2, 3) // thermal constant 2 (INVALID) Method(_TPT, 1) { Store (Arg0, _SB.CPU0.TPT)} // trip point temp Method(_TST) { Return (_SB.CPU0.TST) } // temp sensor threshold } // end of CPU0 scope } // end of SSDT // // CPU1 3.0 Thermal Model SSDT // DefinitionBlock ("CPU1SSDT.aml", "OEM1", 0x01, "PmRef", "CPU1", 0x3000) { External(_SB.CPU1, DeviceObj) External(_SB.PCI0.ISA0.TZ0, ThermalZoneObj) Scope(_SB.CPU1) { // // Add the objects required for 3.0 extended thermal support // // Create a region and fields for thermal support; the platform // fills in the values and traps on writes to enable hysteresis. // The Operation Region location is invalid OperationRegion(CP01, SystemIO, 0x00000008, 0xA) Field(CP01, ByteAcc, Lock, Preserve) { SCP, 1, // thermal policy (passive/active) RTV, 1, // absolute or relative temperature , 6, // reserved AC0, 16, // active cooling temp PSV, 16, // passive cooling temp CRT, 16, // critical temp TPT, 16, // Temp trip point crossed TST, 8 // Temp sensor threshold } Method(_TZM, 0) { Return(_SB.PCI0.ISA0.TZ0) } // thermal zone member // Some thermal zone methods are now located under the // thermal device participating in the 3.0 thermal model. // These methods provide device specific thermal information Method(_SCP, 1) { Store (Arg0, _SB.CPU1.SCP) } // set cooling mode Method(_RTV) { Return (_SB.CPU1.RTV) } // absolute or relative temp Method(_AC0) { Return (_SB.CPU1.AC0) } // active cooling (fan) temp Method(_PSV) { Return (_SB.CPU1.PSV) } // passive cooling temp Method(_CRT) { Return (_SB.CPU1.CRT) } // critical temp Name(_TC1, 4) // thermal constant 1 (INVALID) Name(_TC2, 3) // thermal constant 2 (INVALID) Method(_TPT, 1) { Store (Arg0, _SB.CPU1.TPT)} // trip point temp Method(_TST) { Return (_SB.CPU1.TST) } // temp sensor threshold } // end of CPU1 scope } // end of SSDT © Copyright 2022, UEFI Forum, Inc. Built with Sphinx using a theme provided by Read the Docs.