Server Base Boot Requirements System Software on ARM® Platforms

Document number: ARM DEN 0044B

Server Base Boot Requirements
System Software on ARM

Release information

The Change History table lists the changes made to this document.

Table 1-1 Change history

Date	Issue	Confidentiality	Change
16 August 2014	A	Non-Confidential	Initial release, SBBR version 0.9
8 March 2016	B	Non-Confidential	SBBR version 1.0 Updated referenced specifications to: UEFI 2.5, ACPI 6.0, SMBIOS 3.0.0

1 ABOUT THIS DOCUMENT

1.1 Introduction

This Server Base Boot Requirements (SBBR) specification is intended for SBSA[2]-compliant 64-bit ARMv8 servers. It defines the base firmware requirements for out-of-box support of any ARM SBSA-compatible Operating System or hypervisor. The requirements in this specification are expected to be minimal yet complete for booting a multi-core ARMv8 server platform, while leaving plenty of room for OEM or ODM innovations and design details.

This specification is intended to be OS-neutral. It leverages the prevalent industry standard firmware specifications of UEFI and ACPI. Several changes in support of ARMv8 platforms have been submitted to the UEFI working groups of the *ARM Bindings Sub Team* (ABST) and the *ACPI Specification Working Group* (ASWG) for review, inclusion, and publication.

1.2 References

This document refers to the following documents.

Reference	Doc No	Authors	Title
[1]	ARM DDI 0487	ARM	ARM® Architecture Reference Manual, ARMv8, for ARMv8-A architecture profile
[2]	ARM DEN 0029	ARM	Server Based System Architecture SBSA Version 2.3
[3]	ACPI 6.0	UEFI.org	Advanced Configuration and Power Interface Specification. Revision 6.0
[4]	UEFI Specification 2.5	UEFI.org	Unified Extensible Firmware Interface Specification. Version 2.5
[5]	ARM DEN 022	ARM	Power State Coordination Interface (PSCI) Version 1.0
[6]	SMBIOS Version 3.0.0	DMTF	System Management BIOS (SMBIOS) Reference Specification
[7]	MP Startup for ARM platforms	Microsoft	Multi-processor Startup for ARM Platforms https://acpica.org/sites/acpica/files/MP Startup for ARM platforms.doc

1.2.1 Cross References

This document cross-references sources that are listed in the References section by using the section sign §.

Examples:

ACPI § 5.6.5	-	Reference to the ACPI specification [3] section 5.6.6
UEFI § 6.1	-	Reference to the UEFI specification [4] section 6.1

1.3 Terms and abbreviations

This document uses the following terms and abbreviations.

Term	Meaning
A64	The 64-bit ARM instruction set used in AArch64 state. All A64 instructions are 32 bits.
AArch64 state	The ARM 64-bit Execution state that uses 64-bit general purpose registers, and a 64-bit program counter (PC), Stack Pointer (SP), and exception link registers (ELR). AArch64 Execution state provides a single instruction set, A64.
ACPI	Advanced Configuration and Power Interface.
EFI Loaded Image	An executable image to be run under the UEFI environment, and which uses boot time services.
EL0	The lowest Exception level. The Exception level that is used to execute user applications, in Non-secure state.
EL1	Privileged Exception level. The Exception level that is used to execute Operating Systems, in Non-secure state.
EL2	Hypervisor Exception level. The Exception level that is used to execute hypervisor code. EL2 is always in Non-secure state.
EL3	Secure Monitor Exception level. The Exception level that is used to execute Secure Monitor code, which handles the transitions between Non-secure and Secure states. EL3 is always in Secure state.
OEM	Original Equipment Manufacturer. In this document, the final device manufacturer.
SiP	Silicon Partner. In this document, the silicon manufacturer.
UEFI	Unified Extensible Firmware Interface.
UEFI Boot Services	Functionality that is provided to UEFI Loaded Images during the UEFI boot process.
UEFI Runtime Services	Functionality that is provided to an Operating System after the ExitBootServices() call.

2 SCOPE

This document defines the boot and Runtime Services that are expected by an enterprise platform Operating System or hypervisor, for an ARM AArch64 server, which is SBSA-compliant and follows the UEFI and ACPI specifications.

The UEFI and ACPI specifications have been chosen to ease the adoption of ARM, by aligning the AArch64 server ecosystem to the existing enterprise server market. Many other AArch64 systems exist within other market segments, but their boot and firmware choices are beyond the scope of this document.

This document references the following specification and versions:

UEFI 2.5	Published April 2015, includes the AArch64 bindings.
ACPI 6.0	Published April 2015, includes the Reduced HW profile.

This specification defines the boot and Runtime Services for a physical system, including services that are required for virtualization. It does not define a standardized abstract virtual machine view for a Guest Operating System.

When present with in a system, this document makes additional references to the Power State Coordination Interface:

PSCI 1.0	Published January 2013.

The pre-EFI boot environment is not in the scope of this specification, which includes EL3 Firmware, Secure Operating Systems, and the Trusted Boot Process[1].

[1] The ARM Trusted Firmware project provides example EL3 Firmware:
https://github.com/ARM-software/arm-trusted-firmware

3 UEFI

3.1 UEFI Version

Boot and system firmware for 64-bit ARM servers is based on the UEFI specification[4], version 2.5 or later, incorporating the AArch64 bindings.

3.2 UEFI Compliance

Any UEFI-compliant system must follow the requirements that are laid out in section 2.6 of the UEFI specification. However, to ensure a common boot architecture for server-class AArch64, systems compliant with this specification must always provide the UEFI services and protocols that are listed in Appendix A, Appendix B , and Appendix C of this document.

3.3 UEFI System Environment and Configuration

3.3.1 AArch64 Exception Levels

The resident AArch64 UEFI boot-time environment is specified to “Use the highest 64-bit Non-secure privilege level available”. This level is either EL1 or EL2, depending on whether or not virtualization is used or supported.

Resident UEFI firmware might target a specific Exception level. In contrast, UEFI Loaded Images, such as third-party drivers and boot applications, must not contain any built-in assumptions that they are to be loaded at a given Exception level during boot time. Since they can legitimately be loaded into EL1 or EL2.

3.3.1.1 UEFI Boot at EL2

Most systems are expected to boot UEFI at EL2, to allow for the installation of a hypervisor or a virtualization aware Operating System.

3.3.1.2 UEFI Boot at EL1

Booting of UEFI at EL1 is most likely within a hypervisor hosted Guest Operating System environment, to allow the subsequent booting of a UEFI-compliant Operating System. In this instance, the UEFI boot-time environment can be provided, as a virtualized service, by the hypervisor and not as part of the host firmware.

3.3.2 Additional environment configuration

The UEFI environment must operate in accordance with the UEFI 2.5 specification. The following specific register settings must be in place before the invocation of the UEFI boot environment:

3.3.2.1 System Configuration Register Errata

- The System Configuration Register (either SCTLR_EL1, or SCTLR_EL2) must be set according to UEFI 2.5, with the following alteration:
    - SCTLR_ELx.A=0 (Alignment Not Enforced).

Note: A future revision of the UEFI specification will incorporate these configuration bit settings for SCTLR_ELx.

3.3.3 System Volume Format

Disks to be used by AArch64 UEFI systems for the purposes of booting the Operating System must contain a GPT partition table with an EFI System Partition (as defined by the UEFI specification).

The legacy MBR partitioning scheme is not supported for booting.

3.3.4 EBC

If an EBC interpreter is implemented, then it must produce the `EFI_EBC_PROTOCOL` interface.

3.3.4.1 UEFI Drivers

If a platform supports the inclusion or addition of any device that provides a container for one or more UEFI drivers that are required for the initialization of that device, then the firmware must implement an EBC interpreter.

If the container for the device includes multiple binary types, then the selection order for execution is defined as follows:

1. A64 binary.
2. EBC binary.
3. Emulation of a non-native binary, for example, x86.

3.3.4.2 UEFI Applications

A UEFI application must consist of either A64 binary or EBC binary.

3.4 UEFI Boot Services

3.4.1 Memory Map

The UEFI environment must provide a system memory map, which must include all appropriate devices and memories that are required for booting and system configuration.

All RAM defined by the UEFI memory map must be identity-mapped, which means that virtual addresses must equal physical addresses.

The default RAM allocated attribute must be `EFI_MEMORY_WB`.

All devices and memories that are described by the UEFI Memory Map must be aligned to a 64KB natural alignment so that an Operating System can use 64KB pages with 64KB translation tables.

Note: UEFI is specified to use 4KB pages.

3.4.2 UEFI Loaded Images

UEFI loaded images for AArch64 must be in 64-bit PE/COFF format and must contain only A64 code.

3.4.3 Real-time Clock

The Real-time Clock must be accessible to the UEFI system firmware, so that compliant platforms have a method to monitor time.

3.4.4 Configuration Tables

A UEFI system that complies with this specification must provide the following tables via the EFI Configuration Table:

- `EFI_ACPI_20_TABLE_GUID`
  - The ACPI tables must be at version ACPI 6.0 or later with a *HW-Reduced ACPI model*. See Section 4.
- `SMBIOS3_TABLE_GUID`
  - This table defines the 64-bit entry point for SMBIOS table.
  - The SMBIOS v3.0 tables must conform to version 3.0.0 or later of the SMBIOS Specification. See Section 5.

3.4.5 UEFI Secure Boot (Optional)

UEFI Secure Boot is optional for this specification.

If Secure Boot is implemented, it must conform to the UEFI 2.5 specification for Secure Boot. There are no additional requirements for Secure Boot.

3.5 UEFI Runtime Services

UEFI Runtime Services exist after the call to ExitBootServices() and are designed to provide a limited set of persistent services to the platform Operating System or hypervisor.

The Runtime Services that are listed in Appendix B must be provided.

3.5.1 Runtime Exception Level

UEFI 2.5 enables runtime services to be supported at either EL1 or EL2, with appropriate virtual address mappings. When called, subsequent runtime service calls must be from the same Exception level.

3.5.2 Runtime Memory Map

Before calling ExitBootServices(), the final call to GetMemoryMap() returns a description of the entire UEFI memory map, that includes the persistent Runtime Services mappings.

After the call to ExitBootServices(), the Runtime Services page mappings can be relocated in virtual address space by calling SetVirtualAddressMap(). This call allows the Runtime Services to assign virtual addresses that are compatible with the incoming Operating System memory map.

A UEFI runtime environment compliant with this specification must not be written with any assumption of an identity mapping between virtual and physical memory maps.

UEFI operates with a 4K page size. With Runtime Services, these pages are mapped into the Operating System address space.

To allow Operating Systems to use 64K page mappings, UEFI 2.5, constrains all mapped 4K memory pages to have identical page attributes, within the same physical 64K page.

3.5.3 Real-time Clock

The Real-time Clock must be accessible via the UEFI runtime firmware, and the following services must be provided:

- GetTime().
- SetTime().

It is permissible for SetTime() to return an error on systems where the Real-time Clock cannot be set by this call.

3.5.4 UEFI Reset and Shutdown

The UEFI Runtime service ResetSystem() must implement the following commands, for purposes of power management and system control.

- EfiResetCold.
- EfiResetShutdown.
    - EfiResetShutdown must not reboot the system.

If firmware updates are supported through the Runtime Service of UpdateCapsule(), then ResetSystem() might need to support the following command:

- EFiWarmReset.

If PSCI is present, these Runtime Services must be implemented by calling into PSCI.

Note: When Runtime Services and PSCI co-exist, it is anticipated that Operating System calls to reset the system will go via Runtime Services and not directly to PSCI.

3.5.5 Set Variable

Non-volatile UEFI variables must persist across reset, and emulated variables in RAM are not permitted.

The UEFI Runtime Services must be able to update the variables directly without the aid of the Operating System.

Note: This normally requires dedicated storage for UEFI variables that is not directly accessible from the Operating System.

4 ACPI REQUIREMENTS

SBSA-compliant servers use the *Advanced Configuration and Power Interface* (ACPI) to describe the hardware resources that are installed, and to handle aspects of runtime system configuration, event notification, and power management.

The Operating System must be able to use ACPI to configure the platform hardware and provide basic operations. The ACPI tables are passed, via UEFI, into the Operating System to drive the OSPM (Operating System-directed Power Management).

This section defines mandatory and optional ACPI features, and a few excluded features.

4.1 ACPI Provided Data Structures

All platforms compliant with this specification must:

- Conform to the ACPI specification, version 6.0 or later.
    - Legacy tables and methods are not supported.
- Implement the HW-Reduced ACPI model. See ACPI § 3.11.1 and 4.1.
- Not support legacy ACPI Fixed Hardware interfaces.
- Provide *GPIO-Signaled Events* (see ACPI § 5.6.5) for the conveyance of runtime event notifications, from the system firmware to the *Operating System Power Management* (OSPM).

4.2 ACPI Tables

ACPI tables are essentially data structures. The OSPM of the Operating System receives a pointer to the *Root System Description Pointer* (RSDP) from the boot loader. The OSPM then uses the information in the RSDP to determine the addresses of all other ACPI tables. The ACPI tables might be stored in ROM or flash memory as decided by the platform designer.

All platforms compliant with this specification:

- Must ensure that the structure of all tables is consistent with the ACPI 6.0 or later specification.
    - Legacy tables are not supported.
- Must ensure that the pointer to the RSDP is passed via UEFI to the OSPM as described by UEFI.
- Must use 64-bit addresses within all address fields in ACPI tables.
    - This restriction ensures a long-term future for the ACPI tables. Versions before ACPI 5.0 allowed 32-bit address fields.

4.2.1 Mandatory ACPI Tables

The following tables are mandatory for all compliant systems.

4.2.1.1 RSDP

- *Root System Description Pointer* (RSDP), ACPI § 5.2.5.
    - Within the RSDP, the RsdtAddress field must be null (zero) and the XsdtAddresss MUST be a valid, non-null, 64-bit value.

4.2.1.2 XSDT

- *Extended System Description Table* (XSDT), ACPI § 5.2.8.
    - The RSDP must contain a pointer to this table.
    - This table in turn contains pointers to all other ACPI tables that are to be used by the OSPM.

4.2.1.3 FADT

- *Fixed ACPI Description Table* (FADT), ACPI § 5.2.9
    - The ACPI signature for this table is actually *FACP*. The name *FADT* is used for historical reasons.
    - This table must have the HW_REDUCED_ACPI flag set to comply with the *HW-Reduced ACPI* model. Many other fields must be set to null when this flag is set.
    - It is recommended that one of the server profiles (ACPI § 5.2.9.1) be selected.
    - The ARM_BOOT_ARCH flags describe the presence of PSCI. See ACPI § 5.2.9.4.

4.2.1.4 DSDT and SSDT

- *Differentiated System Description Table* (DSDT), ACPI § 5.2.11.1.
    - This table provides the essential configuration information that is needed to boot the platform.
- *Secondary System Description Table* (SSDT), ACPI § 5.2.11.2.
    - Additional definition blocks can be provided through SSDT tables.

4.2.1.5 MADT

- *Multiple APIC Description Table* (MADT), ACPI § 5.2.12.
    - This table describes the GIC interrupt controllers, their version, and their configuration.
    - For systems without PSCI, this table provides the *Parked Address* for secondary CPU initialization.

4.2.1.6 GTDT

- *Generic Timer Descriptor Table* (GTDT), ACPI § 5.2.24.
    - This table describes the ARM Generic Timer block and the SBSA watchdog.

4.2.1.7 DBG2

- *Debug Port Table 2* (DBG2). See http://msdn.microsoft.com/en-us/library/windows/hardware/dn639131(v=vs.85).aspx.
    - This table provides a standard debug port.
    - This table describes the ARM SBSA Generic UART.

4.2.1.8 SPCR

- *Serial Port Console Redirection* (SPCR). See http://msdn.microsoft.com/en-us/windows/hardware/gg487465

    - This table provides the essential configuration information that is needed for headless operations, such as a kernel shell or console.
    - This table defines a Serial Port type, location, and interrupts.
- This specification requires a minimal revision 2 of the SPCR table; revisions before 2 are not supported.
- The SPCR must be populated with correct ACPI GSIV interrupt routing information for the UART device.
- The SPCR console device must be included in the DSDT.

4.2.2 Recommended ACPI Tables

ACPI tables that are recommended are listed in Appendix E .

Not every platform that is compliant with this specification provides all of these tables, because many tables reference optional platform features.

Examples:

- While the SPMI table is recommended, not all platforms support IPMIv2 (or later).
- A platform does not have to implement NUMA for memory.
  If it does, however, it must provide the SRAT and SLIT that describe the NUMA topology to ACPI.

4.2.3 Optional ACPI Tables

All other tables that are defined in the ACPI 6.0 specification can be used as needed for AArch64 platforms, if and only if they comply with syntax and semantics of the specification.

4.3 ACPI Definition Blocks

Within the DSDT or SSDT, tables that are used to describe the platform, devices are defined by ACPI *definition blocks* (see ACPI § 5.2.11). Each of these definition blocks describes one or more devices that cannot be enumerated by the OSPM at boot time without additional information. For example, processors must be described by definition blocks, whereas PCI devices are enumerated by a defined protocol.

4.4 ACPI Methods and Objects

Within a DSDT or SSDT definition block are definitions of objects and methods that can be invoked. These definitions can provide global information, but most of them provide information that is specific to a single device. Objects and methods can also be predefined, that is, they are defined either by the ACPI specification or as needed by a platform designer.

All objects and methods must conform to the definitions in ACPI 6.0 or higher, legacy definitions are not supported.

4.4.1 Global Methods and Objects

Platforms must define processors as devices under the _SB (System Bus) namespace. See ACPI § 5.3.1

Platforms must not define processors using the global _PR (Processors) namespace. See ACPI § 5.3.1

Platforms compliant with this specification must provide the following predefined global methods:

- _ _SST: System Status Indicator. This method reports on the current overall state of the system status indicator, if and only if a platform provides a user-visible status such as an LED.
    - See ACPI § 9.1.1

- _OSI: Operating System Interfaces. This method defines what interfaces are available to ACPI or the OSPM.
    - See ACPI § 5.7.2

Note: _OSI and _REV are provided by the OSPM implementation and not the platform data.

Note: The long-term trend is to replace the need for _OSI with feature-specific _OSC calls.

4.4.2 Device Methods and Objects

For each device definition in the platform DSDT and SSDT tables, platforms must provide the following predefined methods or objects in accordance with their definitions in version 6.0 or later of the ACPI specification:

- _ADR: Address on the parent bus of the device. Either this object or the _HID must be provided. This object is essential for PCI, for example.
    - See ACPI § 6.1.1

- _CCA: Cache Coherency Attribute. This object provides information about whether a device has to manage cache coherency and about hardware support. It is mandatory for all devices that are not cache-coherent, and recommended for all devices.
    - See ACPI § 6.2.17

- _CRS: Current Resource Settings. This method provides essential information to describe resources, such as registers and their locations that are provided by the device.
    - See ACPI § 6.2.2

- _HID: Hardware ID. This object provides the Plug and Play Identifier or the ACPI ID for the device. Either this object or the _ADR must be provided.
    - See ACPI § 6.1.5.

- _STA: Status. This method identifies whether the device is on, off, or removed.
    - See ACPI § 6.3.7 and 7.1.4.

- _UID: Unique persistent ID. This object provides a unique value that is persistent across boots and can uniquely identify the device with either a common _HID or _CID. The object is used, for example, to identify a PCI root bridge, if there are multiple PCI root bridges in the system.
    - See ACPI § 6.1.12.

Note: A _HID object must be used to describe any device that is enumerated by OSPM. OSPM only enumerates a device when no bus enumerator can detect the ID. For example, devices on an ISA bus are enumerated by OSPM. Use the _ADR object to describe devices that are enumerated by bus enumerators other than OSPM.

4.4.3 GPIO Controllers

The HW-Reduced ACPI model has specific requirements for GPIO controllers and devices. Platforms compliant with this specification must provide the following methods:

- _AEI: ACPI Event Interrupts. This object defines which GPIO interrupts are to be handled as ACPI events.
    - See ACPI § 5.6.5.2.
- _EVT: Event method for GPIO-signaled interrupts. For event numbers less than 255, the _Exx or _Lxx methods can be used instead.
    - See ACPI § 5.6.5.3 and 5.6.4.1.

Embedded controllers (EC) can report to any GPIO interrupt on a HW-Reduced platform.

4.5 Hardware Requirements Imposed on the Platform by ACPI

The term *HW-Reduced* does not imply anything about functionality. HW-Reduced simply means that the hardware specification is not implemented, see Chapter 4 of the ACPI specification. All functionality is still supported, through equivalent software-defined interfaces.

What is reduced is the complexity of the OSPM in supporting ACPI. For example, many requirements from versions earlier than version 5.0 can be ignored. At the same time, this model does impose some requirements on the hardware that is provided by the platform. In particular, ACPI 6.0 relies on *GPIO-Signaled* events. See

ACPI § 5.6.5. This implies that platforms compliant with this specification must have *GPIO lines* that can generate interrupts, and that those interrupts are functionally equivalent to *General Purpose Events* (GPEs), see ACPI § 5.6.4.

Platforms compliant with this specification must provide the following *GPIO-Signaled* platform events:

- For the *ACPI Platform Error Interface* (APEI):
    - One event for non-fatal error signaling (ACPI § 18.3.2.6.2).
    - One NMI-equivalent signal for use in fatal errors.
    - See ACPI § 18.
- For each *Embedded Controller* (EC) on the platform, at least one defined interrupt.
    - See ACPI § 9.6, 12, 4.1.1.1, 4.8.4.2.2, and especially 12.6.
- At least one wake signal, which is routed via a platform event.

4.5.1 Platform Communication Channel (PCC)

This specification highly recommends that the Platform Communications Channel be used. See ACPI § 14.

Note: An interrupt must be defined specifically for the doorbell.

This specification recommends that the *Collaborative Processor Performance Control* (CPPC) be used.

It also recommends use of the *RAS Feature Table* (RASF) and the *Memory Power State Table* (MPST) when appropriate.

4.5.2 Time and Alarm Device

If the ACPI Time and Alarm Device is implemented, see ACPI § 9.18, it must operate on the same real-time clock that is exposed by the UEFI Runtime Services.

5 SMBIOS

Published by the DMTF, the *System Management BIOS* (SMBIOS) is an important firmware component for servers. SMBIOS provides basic hardware and firmware configuration information through table-driven data structures. Although it is not required for Operating System booting or core kernel functions, SMBIOS is widely used for platform management, scripting, and deployment applications.

5.1 SMBIOS Base Requirements

The SMBIOS table is required to conform to the SMBIOS specification, version 3.0.0 or later. Legacy SMBIOS tables and formats are not supported.

5.1.1 SMBIOS requirements on UEFI

- UEFI uses SMBIOS3_TABLE_GUID to identify the SMBIOS table.
- UEFI uses the EfiRuntimeServicesData type for the system memory region containing the SMBIOS table.
- UEFI must not the use the EfiBootServicesData type for the SMBIOS data region, as the region could be reclaimed by a UEFI-compliant Operating System after UEFI ExitBootServices() is called.

5.2 SMBIOS Structures

SMBIOS implementations vary by server design and form-factor. For an SBBR-compliant server, the tables and information that are described in the following subsections are either required or recommended.

5.2.1 Type00: BIOS Information (REQUIRED)

- Vendor.
- BIOS Version.
- BIOS Release Date.
- BIOS ROM Size.
- System BIOS Major Release.
- System BIOS Minor Release.
- Embedded Controller Firmware Major Release.
- Embedded Controller Firmware Minor Release.

5.2.2 Type01: System Information (REQUIRED)

- Manufacturer
- Product Name
- Version
- Serial Number
- UUID
- SKU Number

5.2.3 Type02: Baseboard (or Module) Information (RECOMMENDED)

- Manufacturer

- Product
- Version
- Serial Number
- Asset Tag
- Location in Chassis
- Board Type

5.2.4 Type03: System Enclosure or Chassis (REQUIRED)

- Manufacturer
- Type
- Version
- Serial Number
- Asset Tag Number
- Height
- SKU Number

5.2.5 Type04: Processor Information (REQUIRED)

- Socket Designation
- Processor Type
- Processor Family
- Processor Manufacturer
- Processor Version
- Max Speed
- Status
- Core Count
- Core Enabled
- Thread Count
- Processor Family 2
- Core Count 2
- Core Enabled 2
- Thread Count 2

Exactly one Type4 structure must be provided for every socket in the system, for example, N Type4 structures, in a one-to-one mapping with each physical socket, out of a socket count of N.

- A physical socket is defined as a discrete SoC or equivalent physical chip package implementing a chip-to-chip extension of cache coherency and typically participating within the same Inner Shareable domain, as defined in[1].

5.2.6 Type07: Cache Information (REQUIRED)

- Socket Designation
- Cache Configuration
- Maximum Cache Size
- Installed Size
- Cache Speed

5.2.7 Type08: Port Connector Information (RECOMMENDED for platforms with physical ports)

- Internal Reference Designator
- Internal Connector Type
- External Reference Designator
- External Connector Type
- Port Type

5.2.8 Type09: System Slots (REQUIRED for platforms with expansion slots)

- Slot Designation
- Slot Type
- Slot Data Bus Width
- Current Usage
- Slot ID
- Slot Characteristics 1
- Slot Characteristics 2
- Segment Group Number
- Bus Number
- Device Function Number

5.2.9 Type11: OEM Strings (RECOMMENDED)

- Count

5.2.10 Type13: BIOS Language Information (RECOMMENDED)

- Installable Languages
- Flags
- Current Language

5.2.11 Type15: System Event Log (RECOMMENDED)

5.2.12 Type16: Physical Memory Array (REQUIRED)

- Location
- Use
- Maximum Capacity
- Number of Memory Devices
- Extended Maximum Capacity

5.2.13 Type17: Memory Device (REQUIRED)

- Total Width

- Data Width
- Size
- Device Locator
- Memory Type
- Type Detail
- Speed
- Manufacturer
- Serial Number
- Asset Tag
- Part Number
- Extended Size

5.2.14 Type19: Memory Array Mapped Address (REQUIRED)

- Starting Address
- Ending Address
- Extended Starting Address
- Extended Ending Address

5.2.15 Type32: System Boot Information (REQUIRED)

- Boot Status

5.2.16 Type38: IPMI Device Information (REQUIRED for platforms with IPMI BMC Host Interface)

- IPMI Specification Revision
- I2C Slave Address
- Base Address
- Base Address Modifier
- Interrupt Info
- Interrupt Number

5.2.17 Type41: Onboard Devices Extended Information (RECOMMENDED)

- Reference Designation
- Device Type
- Device Type Instance
- Segment Group Number
- Bus Number
- Device Function Number

6 SECONDARY CORE BOOT

UEFI is defined as a uniprocessor specification that only uses a single CPU core for booting.
Within modern ARM systems, there are two methods to bring a secondary core online.

1. PSCI [5] – Power State Co-ordination Interface.
   1. This method is described to the platform by two bits within the FADT, see the FADT.PSCI bits.
   2. All secondary cores remain powered down during boot.
   3. After boot, OSPM can call CPU_ON() into the PSCI firmware to power up a chosen core.
   4. The PSCI firmware powers up, initializes the core, and starts execution at the provided address.

2. MP Start-up for ARM [7]. This method is also described as a CPU Parking Protocol.
   1. MP Start-up protocol is described to the platform by the Parking Protocol Version number, within the GICC structure within the MADT.
   2. During the boot process, all secondary cores are powered up and booted into a holding pen, where they remain parked.
   3. The parked cores wait for an interrupt and look for a non-zero start address into their mailbox.
   4. After boot, OSPM can start a secondary core by writing to the mailbox and interrupting the core.
   5. On receipt of both the start address and the interrupt, the core PE continues booting from the start address.

6.1 Secondary Core Boot Requirements

The chosen long-term direction for secondary core booting is to align on PSCI. Until that time, the MP Start-up and PSCI methods co-exist.

- When PSCI is present in the platform, it must be the method for secondary boot.
- When PSCI is not present, MP Start-up must be presented by boot firmware.
- Compliant Operating Systems must support PSCI and MP Start-up.

Note: During the transition to full SBBR compliance, server firmware may provide the mixed protocols of MP Start-up and PSCI. This will allow a legacy MP Start-up only Operating System to boot on a PSCI-based system. How to configure firmware to provide the mixed protocols is implementation defined.

It is recommended that Operating Systems gracefully handle the mixed case of MP Start-up and PSCI.

The following table lists the secondary core boot permutations:

Method	FADT.PSCI	MADT.GICC.PPV	Hand over state of secondary cores
PSCI	1	0	Secondary cores are powered down.
MP Start-up	0	> 0	Secondary cores are powered up before handing to the OS.
Mixed	1	> 0	Secondary cores that are powered up (via PSCI) before handing to the OS.
None	0	0	No secondary boot is specified.

MADT.GICC.PPV is the parking protocol version number. A value of 0 identifies a PSCI exclusive system.

APPENDIX A REQUIRED UEFI BOOT SERVICES

Service	UEFI §
EFI_RAISE_TPL	6.1
EFI_RESTORE_TPL	6.1
EFI_ALLOCATE_PAGES	6.2
EFI_FREE_PAGES	6.2
EFI_GET_MEMORY_MAP	6.2
EFI_ALLOCATE_POOL	6.2
EFI_FREE_POOL	6.2
EFI_CREATE_EVENT	6.1
EFI_SET_TIMER	6.1
EFI_WAIT_FOR_EVENT	6.1
EFI_SIGNAL_EVENT	6.1
EFI_CLOSE_EVENT	6.1
EFI_FREE_POOL	6.1
EFI_INSTALL_PROTOCOL_INTERFACE	6.3
EFI_REINSTALL_PROTOCOL_INTERFACE	6.3
EFI_UNINSTALL_PROTOCL_INTERFACE	6.3
EFI_HANDLE_PROTOCOL	6.3
EFI_REGISTER_PROTOCOL_NOTIFY	6.3
EFI_LOCATE_HANDLE	6.3
EFI_LOCATE_PROTOCOL	6.3
EFI_LOCATE_DEVICE_PATH	6.3
EFI_INSTALL_CONFIGURATION_TABLE	6.3
EFI_IMAGE_LOAD	6.4
EFI_IMAGE_START	6.4
EFI_EXIT	6.4
EFI_IMAGE_UNLOAD	6.4
EFI_EXIT_BOOT_SERVICES	6.4
EFI_GET_NEXT_MONOTONIC_COUNT	6.5
EFI_STALL	6.5
EFI_SET_WATCHDOG_TIMER	6.5
EFI_CONNECT_CONTROLLER	6.3
EFI_DISCONNECT_CONTROLLER	6.3
EFI_OPEN_PROTOCOL	6.3
EFI_CLOSE_PROTOCOL	6.3
EFI_OPEN_PROTOCOL_INFORMATION	6.3
EFI_PROTOCOLS_PER_HANDLE	6.3
EFI_LOCATE_HANDLE_BUFFER	6.3
EFI_LOCATE_PROTOCOL	6.3
EFI_INSTALL_MULTIPLE_PROTOCOL_INTERFACES	6.3
EFI_UNINSTALL_MULTIPLE_PROTOCOL_INTERFACES	6.3
EFI_CALCULATE_CRC32	6.5
EFI_COPY_MEM	6.5
EFI_SET_MEM	6.5
EFI_CREATE_EVENT_EX	6.5

APPENDIX B REQUIRED UEFI RUNTIME SERVICES

Service	UEFI §
EFI_GET_TIME	7.3
EFI_SET_TIME	7.3
EFI_GET_WAKEUP_TIME	7.3
EFI_SET_WAKEUP_TIME	7.3
EFI_SET_VIRTUAL_ADDRESS_MAP	7.4
EFI_CONVERT_POINTER	7.4
EFI_GET_VARIABLE	7.2
EFI_GET_NEXT_VARIABLE_NAME	7.2
EFI_SET_VARIABLE	7.2
EFI_UPDATE_CAPSULE	7.5
EFI_QUERY_CAPSULE_CAPABILITIES	7.5
EFI_QUERY_VARIABLE_INFO	7.5

Note: EFI_GET_WAKEUP_TIME and EFI_SET_WAKEUP_TIME must be implemented, but might simply return EFI_UNSUPPORTED.

UEFI Configuration Table Entries

Configuration Table
EFI_ACPI_20_TABLE_GUID
SMBIOS3_TABLE_GUID

APPENDIX C REQUIRED UEFI PROTOCOLS

Core UEFI Protocols

Service	UEFI §
EFI_LOADED_IMAGE_PROTOCOL	8.1
EFI_LOADED_IMAGE_DEVICE_PATH_PROTOCOL	8.1
EFI_DECOMPRESS_PROTOCOL	18.5
EFI_DEVICE_PATH_UTILITIES_PROTOCOL	9.3

Media I/O Protocols

Service	UEFI §
EFI_LOAD_FILE_PROTOCOL	12.1
EFI_LOAD_FILE2_PROTOCOL	12.2
EFI_SIMPLE_FILE_SYSTEM_PROTOCOL	12.4
EFI_FILE_PROTOCOL	12.5

Console Protocols

Service	UEFI §
EFI_SIMPLE_TEXT_INPUT_PROTOCOL	11.2
EFI_SIMPLE_TEXT_INPUT_EX_PROTOCOL	11.3
EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL	11.4

Driver Configuration Protocols

Service	UEFI §
EFI_HII_DATABASE_PROTOCOL	29.4
EFI_HII_STRING_PROTOCOL	29.4
EFI_HII_CONFIG_ROUTING_PROTOCOL	29.4
EFI_HII_CONFIG_ACCESS_PROTOCOL	29.4

APPENDIX D OPTIONAL UEFI PROTOCOLS

Basic Networking Support

Service	UEFI §
EFI_SIMPLE_NETWORK_PROTOCOL	21.1
EFI_MANAGED_NETWORK_PROTOCOL	22.1
EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL	22.1

Networking services are optional on platforms that do not support networking.

Network Boot Protocols

Service	UEFI §
EFI_PXE_BASE_CODE_PROTOCOL	23.1
EFI_PXE_BASE_CODE_CALLBACK_PROTOCOL	21.3
EFI_BIS_PROTOCOL	21.5
EFI_MTFTP4_PROTOCOL	26.3
EFI_MTFTP6_PROTOCOL	26.4

EFI_BIS_PROTOCOL is optional on machines that do not support Secure Boot.

Ipv4 Network Support

Service	UEFI §
EFI_ARP_PROTOCOL	25.1
EFI_ARP_SERVICE_BINDING_PROTOCOL	25.1
EFI_DHCP4_SERVICE_BINDING_PROTOCOL	25.2
EFI_DHCP4_PROTOCOL	25.2
EFI_TCP4_PROTOCOL	24.1.3
EFI_TCP4_SERVICE_BINDING_PROTOCOL	24.1.1
EFI_IP4_SERVICE_BINDING_PROTOCOL	23.3.1
EFI_IP4_CONFIG_PROTOCOL	23.3.3
EFI_UDP4_PROTOCOL	26.1.2

EFI_UDP4_SERVICE_BINDING_PROTOCOL	26.1.1

Networking services are optional on platforms that do not support networking.

Ipv6 Networking Support

Service	UEFI §
EFI_DHCP6_PROTOCOL	25.3.3
EFI_DHCP6_SERVICE_BINDING_PROTOCOL	25.3.1
EFI_TCP6_PROTOCOL	24.5.3
EFI_TCP6_SERVICE_BINDING_PROTOCOL	24.2.1
EFI_IP6_SERVICE_BINDING_PROTOCOL	24.5.1
EFI_IP6_CONFIG_PROTOCOL	26.1
EFI_UDP6_PROTOCOL	26.2.3
EFI_UDP6_SERVICE_BINDING_PROTOCOL	26.2.1

Networking services are optional on platforms that do not support networking.

VLAN Protocols

Service	UEFI §
EFI_VLAN_CONFIG_PROTOCOL	23.1

iSCSI Protocols

Service	UEFI §
EFI_ISCSI_INITIATOR_NAME_PROTOCOL	15.2

Support for iSCSI is only required on machines that lack persistent storage, such as a, HDD. This configuration is intended for thin clients and *compute-only* nodes.

EBC Support

Service	UEFI §
EFI_EBC_PROTOCOL	20.11

If an EBC interpreter is implemented, then the platform must produce the EFI_EBC_PROTOCOL interface.

APPENDIX E RECOMMENDED ACPI TABLES

PCIe

ACPI Signature	Full Name	ACPI §
MCFG	PCI memory-mapped configuration space base address description table	5.2.6

I/O Topology

ACPI Signature	Full Name	ACPI §
IORT	Support for SMMUv2, ITS, and system topology description	5.2.6

Platform Error Interfaces

ACPI Signature	Full Name	ACPI §
BERT	Boot Error Record Table	18.3.1
EINJ	Error Injection Table	18.6.1
ERST	Error Record Serialization Table	18.5
HEST	Hardware Error Source Table	18.3.2

ACPI Platform Error Interfaces (APEI), which convey error information to the Operating System.

RAS

ACPI Signature	Full Name	ACPI §
RASF	RAS Facilities	5.2.20.3

Reliability, Availability, and Serviceability.

IPMI

ACPI Signature	Full Name	ACPI §
SPMI	Server Platform Management Interface Table	IPMIv2

Used for IPMIv2.

NUMA

ACPI Signature	Full Name	ACPI §
SLIT	System Locality Information Table	5.2.17
SRAT	System Resource Affinity Table	5.2.16

Tables to describe topology and resources that are required by NUMA systems.

Core System Resources Table (CSRT)

ACPI Signature	Full Name	ACPI §
CSRT	Core System Resource Table	http://uefi.org/acpi

Device description for non-standard devices.

Embedded Controller Description Table

ACPI Signature	Full Name	ACPI §
ECDT	Embedded Controller Description Table	5.2.15

Memory Power management

ACPI Signature	Full Name	ACPI §
MPST	Memory Power State Table	5.2.21

Platform Communications Channel (PCC)

ACPI Signature	Full Name	ACPI §
PCCT	Platform Communications Channel Table	14

Provides the interface to communicate to an on-platform controller.

APPENDIX F RECOMMENDED ACPI METHODS

CPU performance control

For CPU performance and control, there are two mutually exclusive methods defined.

Method	Full Name	ACPI §
_CPC	Continuous Performance Control (replaces _PCT and _PSS)	8.4.5.1

Newer _CPC method in conjunction with the PCCT

_PSS	Performance Supported States (Superseded by _CPC)	8.4.4.2

Older _PSS method.

CPU and system idle control

For CPU and system idle management, the following method has been introduced in ACPI6.0.

Method	Full Name	ACPI §
_LPI	Low Power Idle States	8.4.4

NUMA

ACPI Signature	Full Name	ACPI §
_PXM	Proximity	6.2.14
_SLI	System Locality Information	6.2.15

IPMI

Method	Full Name	ACPI §
_IFR	IPMIv2: the IPMI Interface Type, if and only if IPMI has been implemented	IPMI spec
_SRV	IPMIv2: the IPMI revision that is supported by the platform, if and only if IPMI has been implemented	IPMI spec

Device Configuration and Control

Method	Full Name	ACPI §
_CLS	Class code (for non-PCI devices that are compatible with PCI drivers)	6.1.3
_CID	Compatible ID	6.1.2
_DSD	Device Specific Data: Provides additional device properties and information. All compliant _DSD UUIDs and associated definitions must be published by the UEFI Forum, see http://www.uefi.org/acpi	6.2.5
_DSM	Device Specific Method (used to convey info to ACPI that it might not currently have a mechanism to describe, see https://lkml.org/lkml/2013/8/20/556)	9.14.1
_INI	Initialize a device	6.5.1

Resources

Method	Full Name	ACPI §
_MLS	Human readable description in multiple languages. **Note**: this is preferred over _STR.	6.1.7
_PRS	Possible Resource Settings	6.2.11
_SRS	Set Resource Settings	6.2.15
_STR	Device description (in a single language). Superseded by _MLS.	6.1.9