Embedded Base Boot Requirements Platform Design Document

Document number: ARM DEN 0064

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

Table 1-1 Change history

Date	Issue	Confidentiality	Change
20 September, 2017	B	Non-Confidential	Confidentiality Change, EBBR version 0.51

Arm Document License

1 ABOUT THIS DOCUMENT

1.1 Introduction

This Embedded Base Boot Requirements (EBBR) specification is intended for ARM embedded devices that want to take advantage of the UEFI technology to separate the firmware and OS development. For example, class-A embedded devices like networking platforms can benefit from a standard interface that supports features such as secure boot and firmware update.

This specification defines the base firmware requirements if UEFI is chosen. The requirements in this specification are expected to be minimal yet complete, while leaving plenty of room for innovations and design details.

This specification is intended to be OS-neutral. It leverages the prevalent industry standard firmware specifications of UEFI.

Comments or change requests can be sent to arm.ebbr-discuss@arm.com.

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]	UEFI Specification 2.7	UEFI.org	Unified Extensible Firmware Interface Specification. Version 2.7
[3]	ARM DEN 022	ARM	Power State Coordination Interface (PSCI) Version 1.1

1.2.1 Cross References

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

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.
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 Operating System or hypervisor, for an ARM embedded device, which follows the UEFI specification.

This document references the following specification and versions:

UEFI 2.7 Published June 2017, includes the AArch64 bindings.

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 1.1 Published April 2017.

3 UEFI

3.1 UEFI Version

Boot and system firmware for ARM embedded devices can be based on the UEFI specification[2], version 2.7 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 embedded-class, 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 System Volume Format

The system firmware must support GPT partitioning.

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`.

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 Configuration Tables

A UEFI system that complies with this specification may provide the additional tables via the EFI Configuration Table. For example, ACPI table or Device Tree table may be needed to support configuration and power management.

3.4.4 UEFI Secure Boot (Optional)

UEFI Secure Boot is optional for this specification.

If Secure Boot is implemented, it must conform to the UEFI 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 2.7 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 2.7, 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.

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.

APPENDIX A REQUIRED UEFI BOOT SERVICES

Service	UEFI §
EFI_RAISE_TPL	7.1
EFI_RESTORE_TPL	7.1
EFI_ALLOCATE_PAGES	7.2
EFI_FREE_PAGES	7.2
EFI_GET_MEMORY_MAP	7.2
EFI_ALLOCATE_POOL	7.2
EFI_FREE_POOL	7.2
EFI_CREATE_EVENT	7.1
EFI_SET_TIMER	7.1
EFI_WAIT_FOR_EVENT	7.1
EFI_SIGNAL_EVENT	7.1
EFI_CLOSE_EVENT	7.1
EFI_INSTALL_PROTOCOL_INTERFACE	7.3
EFI_REINSTALL_PROTOCOL_INTERFACE	7.3
EFI_UNINSTALL_PROTOCOL_INTERFACE	7.3
EFI_HANDLE_PROTOCOL	7.3
EFI_REGISTER_PROTOCOL_NOTIFY	7.3
EFI_LOCATE_HANDLE	7.3
EFI_LOCATE_PROTOCOL	7.3
EFI_LOCATE_DEVICE_PATH	7.3
EFI_INSTALL_CONFIGURATION_TABLE	7.3
EFI_IMAGE_LOAD	7.4
EFI_IMAGE_START	7.4
EFI_EXIT	7.4
EFI_IMAGE_UNLOAD	7.4
EFI_EXIT_BOOT_SERVICES	7.4
EFI_GET_NEXT_MONOTONIC_COUNT	7.5
EFI_STALL	7.5
EFI_SET_WATCHDOG_TIMER	7.5
EFI_CONNECT_CONTROLLER	7.3
EFI_DISCONNECT_CONTROLLER	7.3
EFI_OPEN_PROTOCOL	7.3
EFI_CLOSE_PROTOCOL	7.3
EFI_OPEN_PROTOCOL_INFORMATION	7.3
EFI_PROTOCOLS_PER_HANDLE	7.3
EFI_LOCATE_HANDLE_BUFFER	7.3
EFI_LOCATE_PROTOCOL	7.3
EFI_INSTALL_MULTIPLE_PROTOCOL_INTERFACES	7.3
EFI_UNINSTALL_MULTIPLE_PROTOCOL_INTERFACES	7.3
EFI_CALCULATE_CRC32	7.5
EFI_COPY_MEM	7.5
EFI_SET_MEM	7.5
EFI_CREATE_EVENT_EX	7.5

APPENDIX B REQUIRED UEFI RUNTIME SERVICES

Service	UEFI §
EFI_GET_TIME	8.3
EFI_SET_TIME	8.3
EFI_GET_WAKEUP_TIME	8.3
EFI_SET_WAKEUP_TIME	8.3
EFI_SET_VIRTUAL_ADDRESS_MAP	8.4
EFI_CONVERT_POINTER	8.4
EFI_GET_VARIABLE	8.2
EFI_GET_NEXT_VARIABLE_NAME	8.2
EFI_SET_VARIABLE	8.2
EFI_GET_NEXT_HIGH_MONO_COUNT	8.5
EFI_RESET_SYSTEM	8.5
EFI_UPDATE_CAPSULE	8.5
EFI_QUERY_CAPSULE_CAPABILITIES	8.5
EFI_QUERY_VARIABLE_INFO	8.5

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

APPENDIX C REQUIRED UEFI PROTOCOLS

Core UEFI Protocols

Service	UEFI §
EFI_LOADED_IMAGE_PROTOCOL	9.1
EFI_LOADED_IMAGE_DEVICE_PATH_PROTOCOL	9.2
EFI_DECOMPRESS_PROTOCOL	19.5
EFI_DEVICE_PATH_PROTOCOL	10.2
EFI_DEVICE_PATH_UTILITIES_PROTOCOL	10.3

Media I/O Protocols

Service	UEFI §
EFI_LOAD_FILE2_PROTOCOL	13.2
EFI_SIMPLE_FILE_SYSTEM_PROTOCOL	13.4
EFI_FILE_PROTOCOL	13.5

Console Protocols

Service	UEFI §
EFI_SIMPLE_TEXT_INPUT_PROTOCOL	12.2
EFI_SIMPLE_TEXT_INPUT_EX_PROTOCOL	12.3
EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL	12.4

Driver Configuration Protocols

Service	UEFI §
EFI_HII_DATABASE_PROTOCOL	33.4
EFI_HII_STRING_PROTOCOL	33.4
EFI_HII_CONFIG_ROUTING_PROTOCOL	33.4
EFI_HII_CONFIG_ACCESS_PROTOCOL	33.4

APPENDIX D OPTIONAL UEFI PROTOCOLS

Basic Networking Support

Service	UEFI §
EFI_SIMPLE_NETWORK_PROTOCOL	24.1
EFI_MANAGED_NETWORK_PROTOCOL	25.1
EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL	25.1

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

Network Boot Protocols

Service	UEFI §
EFI_PXE_BASE_CODE_PROTOCOL	24.3
EFI_PXE_BASE_CODE_CALLBACK_PROTOCOL	24.4
EFI_BIS_PROTOCOL	24.5
EFI_MTFTP4_PROTOCOL	30.3
EFI_MTFTP6_PROTOCOL	30.4

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

Ipv4 Network Support

Service	UEFI §
EFI_ARP_PROTOCOL	29.1
EFI_ARP_SERVICE_BINDING_PROTOCOL	29.1
EFI_DHCP4_SERVICE_BINDING_PROTOCOL	29.2
EFI_DHCP4_PROTOCOL	29.2
EFI_TCP4_PROTOCOL	28.1.2
EFI_TCP4_SERVICE_BINDING_PROTOCOL	28.1.1
EFI_IP4_SERVICE_BINDING_PROTOCOL	28.3.1
EFI_IP4_CONFIG2_PROTOCOL	28.5
EFI_UDP4_PROTOCOL	30.1.2
EFI_UDP4_SERVICE_BINDING_PROTOCOL	30.1.1

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

Ipv6 Networking Support

Service	UEFI §
EFI_DHCP6_PROTOCOL	29.3.2
EFI_DHCP6_SERVICE_BINDING_PROTOCOL	29.3.1
EFI_TCP6_PROTOCOL	28.2.2
EFI_TCP6_SERVICE_BINDING_PROTOCOL	28.2.1
EFI_IP6_SERVICE_BINDING_PROTOCOL	28.6.1
EFI_IP6_CONFIG_PROTOCOL	28.7
EFI_UDP6_PROTOCOL	30.2.2
EFI_UDP6_SERVICE_BINDING_PROTOCOL	30.2.1

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

VLAN Protocols

Service	UEFI §
EFI_VLAN_CONFIG_PROTOCOL	27.1

iSCSI Protocols

Service	UEFI §
EFI_ISCSI_INITIATOR_NAME_PROTOCOL	16.2

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.