Requirements List of RISC-V UEFI PROTOCOL Specification,

Document Version 1.0[*]

by

Mehmet Öner

Version 1.0

*`RISC-V UEFI PROTOCOL Specification, Document Version 1.0', Contributors Sunil V L <sunilvl@ventanamicro.com>, Ard Biesheuvel <ardb@kernel.org>, Heinrich Schuchardt <heinrich.schuchardt@canonical.com>, Jessica Clarke <jrtc27@jrtc27.com>, Atish Patra <atishp@atishpatra.org>, Anup Patel <apatel@ventanamicro.com>, Abner Chang <abner.chang@hpe.com>, RISC-V International, May 2022.
github.com/riscv-non-isa/riscv-uefi
It is licensed under the Creative Commons Attribution 4.0 International License (CC-BY 4.0).
https://creativecommons.org/licenses/by/4.0/

About

The Document

In systems engineering approach, before doing anything with the design of the system under consideration, *requirements analysis* must be completed as one of the first tasks (if not the very first). Beginning with an itemized, atomic, classified and well defined list of requirements is essential. Because, following activities at various stages of development (like design coverage analysis, testing, verification, validation …) depend on the requirement specifications stated at the beginning.

RISC-V International provides the ISA (Instruction Set Architecture) and non-ISA requirement specifications for the RISC-V architecture (https://riscv.org/technical/specifications/). These documents in general are good written technical plain text documents. However, they lack some aspects of good requirement specification practices:

- Requirements are in free text form and not itemized: Itemized list of requirements enables requirement coverage in design, test, verification and validation phases.
- Text includes comments and information statements along with requirements: Statements must be clearly labeled and categorized.
- Some statements include more than one specifications: Each specification need to be isolated.
- Some specifications such as instruction definitions are distributed throughout the text: The distributed content need to be put together to have a complete the specification.

The aim of this document is providing an edited list of requirements for `RISC-V UEFI PROTOCOL Specification, Document Version 1.0', Contributors Sunil V L, Ard Biesheuvel, Heinrich Schuchardt, Jessica Clarke, Atish Patra, Anup Patel, Abner Chang, RISC-V International, May 2022. github.com/riscv-non-isa/riscv-uefi

In particular:

- Statements were itemized and given an ID number.
- Each itemized statement was referenced to the original document to provide traceability
- Itemized statements were categorized
- Complex statements were broken into simpler atomic requirement statements when needed.
- Distributed requirement information was put together to form complete specifications.

Special attention was given to preserve original statements, even when dividing complex statements into simpler atomic statements. But occasionally, some statements were re-written as to form a formal requirement statement.

The Editor (or the systems engineer)

After 34 years of my career, I retired from my regular job in 2023. Now, I do part time consulting services to interested parties, while I do work on projects that interest me more than a regular work.

In my career I dealt with very diverse fields of engineering: Academics, C/C++ desktop programming, embedded systems, analog circuit design, DSP algorithms, VLSI/FPGA design, underwater acoustics are to name few.

I’ve always enjoyed designing controllers and processors with generic HDL for VLSI or FPGA. So, as my personal project to work on, I decided to design RISC-V cores with different capabilities.

Having some defense sector background, I find systems engineering approach very useful. After reading RISC-V specification documents, I decided to take the initiative and edit the documents into customer requirements list format which is a tough, tedious and time consuming work.

In case someone else could find these documents useful, I share them on github:
https://github.com/vizionerco/RISC-V

Best regards,

Mehmet Öner, Ph.D.

www.linkedin.com/in/mehmet-oner-00453733/

Table of Contents

Definitions

Table 1: Requirement Types

TYPE	NAME	EXPLANATION
H	Heading	Headings in the original document. Headings are included to provide context to the subsequent requirements
I	Information	These are statements that explain some aspects of the subject, but actually do not specify any requirement. For these types, the statement(s) in the text is given as is.
C	Comment	These statements are original comment statements in the RISC_V documentation which are explained as: “Commentary on our design decisions is formatted as in this paragraph. This non-normative text can be skipped if the reader is only interested in the specification itself.” For these types, the statement(s) in the text is given as is.
R	Requirement	These are statements that specify a specific need to be fulfilled. For these types, the statement(s) in the text is given as is where possible. In some cases, information is collected from different tables and figures to form a complete specification. In some cases, context is added in parenthesis to make the requirement self explanatory. In some cases, the statements are broken into single statements requirement statements.
O	Optional Requirement	These are tatements that specify a property that is not mandatory to implement. However if it is chosen to fulfill, it should obey this requirement. Inclusion of the text is the same as R/Requirement type.
T	Tentative Requirement	These are requirements but are not frozen yet by the RISC-V committee. Inclusion of the text is the same as R/Requirement type.

Table 2: Abbreviations & Definitions

SHORT	MEANING
ACPI	Advanced Configuration and Power Interface
DT	Device Tree
EFI	Extensible Firmware Interface
UEFI	Unified Extensible Firmware Interface

CHAPTER 1 Introduction

ID	REFERENCE	TYPE	DEFINITION
RVUE.1.1	1.0 (p.4)	H	Introduction
RVUE.1.2	1.0 (p.4)	I	This specification details all new UEFI protocols required only for RISC-V platforms. These protocol specs are maintained by RISC-V community.

CHAPTER 3 RISCV_EFI_BOOT_PROTOCOL

ID	REFERENCE	TYPE	DEFINITION
RVUE.3.1	3.0 (p.6)	H	RISCV_EFI_BOOT_PROTOCOL
RVUE.3.2	3.0 (p.6)	I	Either Device Tree (DT) or Advanced Configuration and Power Interface (ACPI) configuration tables are used to convey the information about hardware to the Operating Systems.
RVUE.3.3	3.0 (p.6)	I	Some of the information are known only at boot time and needed very early before the Operating Systems/boot loaders can parse the firmware tables.
RVUE.3.4	3.0 (p.6)	I	One example is the boot hartid on RISC-V systems.
RVUE.3.5	3.0 (p.6)	I	On non-UEFI systems, this is typically passed as an argument to the kernel (in `a0`). However, UEFI systems need to follow UEFI application calling conventions and hence it can not be passed in `a0`.
RVUE.3.6	3.0 (p.6)	I	There is an existing solution which uses the /chosen node in DT based systems to pass this information. However, this solution doesn't work for ACPI based systems. Hence, a UEFI protocol is preferred for both DT and ACPI based systems.
RVUE.3.7	3.0 (p.6)	I	This UEFI protocol for RISC-V systems provides early information to the bootloaders or Operating Systems. Firmwares like EDK2 and u-boot need to implement this protocol on RISC-V UEFI systems.
RVUE.3.8	3.0 (p.6)	I	This protocol is typically called by the bootloaders before invoking ExitBootServices(). They then pass the information to the Operating Systems.
RVUE.3.9	3.0 (p.6)	R	The version of RISCV_EFI_BOOT_PROTOCOL specified by this specification is `0x00010000`.
RVUE.3.10	3.0 (p.6)	R	All future revisions must be backwards compatible.
RVUE.3.11	3.0 (p.6)	R	If a new version of the specification breaks backwards compatibility, a new GUID must be defined.
RVUE.3.12	3.1 (p.6)	H	GUID

RVUE.3.13	3.1 (p.6)	R	`#define RISCV_EFI_BOOT_PROTOCOL_GUID \`
			`{ 0xccd15fec, 0x6f73, 0x4eec, \`
			`{ 0x83, 0x95, 0x3e, 0x69, 0xe4, 0xb9, 0x40, 0xbf } }`

RVUE.3.14	3.2 (p.6)	H	Revision Number

RVUE.3.15	3.2 (p.6)	R	`#define RISCV_EFI_BOOT_PROTOCOL_REVISION 0x00010000`
			`#define RISCV_EFI_BOOT_PROTOCOL_LATEST_VERSION \`
			`RISCV_EFI_BOOT_PROTOCOL_REVISION`

RVUE.3.16	3.3 (p.6)	H	Protocol Interface Structure

RVUE.3.17	3.3 (p.6)	R	`typedef struct _RISCV_EFI_BOOT_PROTOCOL {`
			`UINT64 Revision;`
			`EFI_GET_BOOT_HARTID GetBootHartId;`
			`} RISCV_EFI_BOOT_PROTOCOL;`

RVUE.3.18	3.3.1 (p.7)	H	RISCV_EFI_BOOT_PROTOCOL.GetBootHartId
RVUE.3.19	3.4 (p.7)	R	This interface provides the `hartid` of the boot cpu.

ID	REFERENCE	TYPE	DEFINITION
RVUE.3.20	3.4 (p.7)	R	Prototype

```
typedef EFI_STATUS
(EFIAPI *EFI_GET_BOOT_HARTID) (
IN RISCV_EFI_BOOT_PROTOCOL *This,
OUT UINTN *BootHartId
);
```

ID	REFERENCE	TYPE	DEFINITION
RVUE.3.21	3.4 (p.7) Table 1		Parameters (GetBootHartId Parameters)

Parameter	Description
This	Pointer to the protocol
BootHartId	Pointer to the variable receiving the hartid of the boot cpu.

ID	REFERENCE	TYPE	DEFINITION
RVUE.3.22			Status Codes Returned (GetBootHartId Return Value)

Return Value	Description
EFI_SUCCESS	The boot hart id could be returned.
EFI_INVALID_PARAMETER	This parameter is NULL or does not point to a valid RISCV_EFI_BOOT_PROTOCOL implementation.
EFI_INVALID_PARAMETER	BootHartId parameter is NULL.