Requirements List

of

RISC-V Platform-Level Interrupt Controller Specification

Version 1.0.0*

by

Mehmet Öner

Version 1.0

*"RISC-V Platform-Level Interrupt Controller Specification", RISC-V International
https://github.com/riscv/riscv-plic-spec

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 Platform-Level Interrupt Controller Specification", RISC-V International, https://github.com/riscv/riscv-plic-spec

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 statements 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.

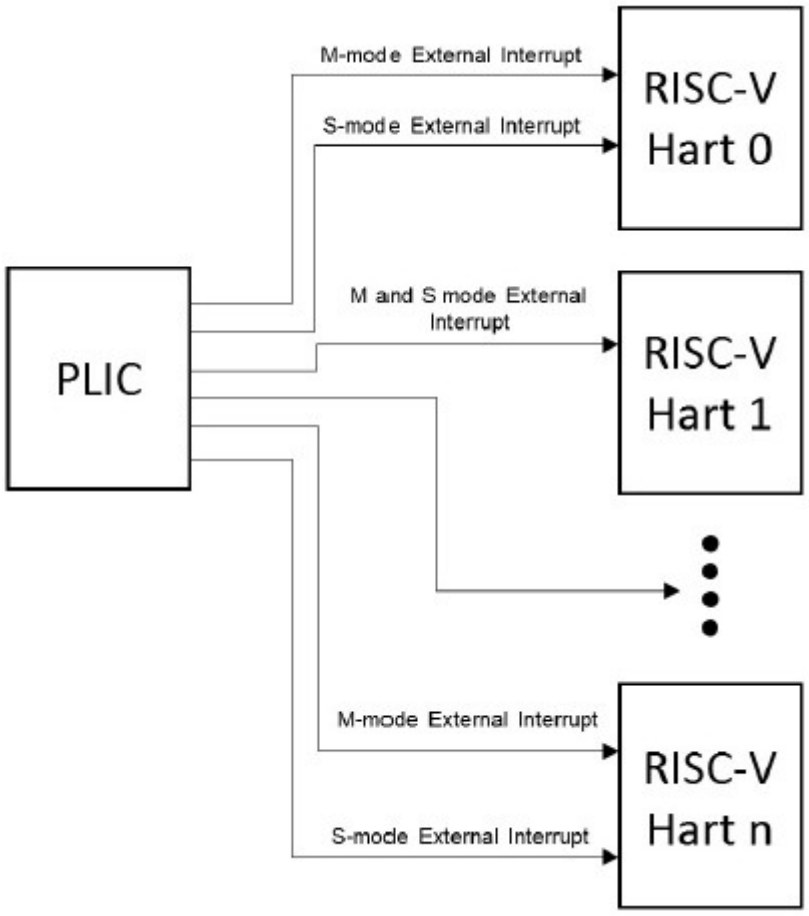
Table 2: Abbreviations & Definitions

SHORT	MEANING
MSI	Message Signaled Interrupt
PLIC	Platform-Level Interrupt Controller
SMT	Simultaneous multithreading

CHAPTER 1 Introduction

ID	REFERENCE	TYPE	DEFINITION
RVIC.1.1	1.0 (p.5)	H	Introduction
RVIC.1.2	1.0 (p.5)	I	This specification delineates the operation parameters according the general PLIC architecture defined in the RISC-V platform-level interrupt controller (PLIC) specification (was removed from RISC-V Privileged Spec v1.11-draft) to work in the context of RISC-V systems.
RVIC.1.3	1.0 (p.5)	R	The PLIC multiplexes various device interrupts onto the external interrupt lines of Hart contexts, with hardware support for interrupt priorities.
RVIC.1.4	1.0 (p.5)	R	PLIC supports up-to 1023 interrupts (0 is reserved) and 15872 contexts, ...
RVIC.1.5	1.0 (p.5)	O	… but the actual number of interrupts and context depends on the PLIC implementation.
RVIC.1.6	1.0 (p.5)	R	However, the implementation must adhere to the offset of each register within the PLIC operation parameters.
RVIC.1.7	1.0 (p.5)	R	The PLIC which claimed as PLIC-Compliant standard PLIC should follow the implementations mentioned in sections below.
RVIC.1.8	1.1 (p.5)	H	Interrupt Targets and Hart Contexts
RVIC.1.9	1.1 (p.5)	I	Interrupt targets are usually hart contexts, where a hart context is a given privilege mode on a given hart (though there are other possible interrupt targets, such as DMA engines).
RVIC.1.10	1.1 (p.5)	I	For example, in an 4-core system with 2-way SMT, you have 8 harts and probably at least two privilege modes per hart: machine mode and supervisor mode.
RVIC.1.11	1.1 (p.5)	I	Not all hart contexts need be interrupt targets, in particular, if a processor core does not support delegating external interrupts to lower-privilege modes, then the lower-privilege hart contexts will not be interrupt targets.
RVIC.1.12	1.1 (p.5)	I	Interrupt notifications generated by the PLIC appear in the `meip`/`seip` bits of the `mip`/`sip` registers for M/S modes respectively.
RVIC.1.13	1.1 (p.5)	C	Previous versions of this specification indicated that the PLIC supports U-mode interrupts. This text was removed because the privileged architecture does not define U-mode interrupts. If a future privileged architecture specifies U-mode interrupts, this PLIC specification can be straightforwardly extended to support them.
RVIC.1.14	1.1 (p.5)	R	The notification only appear in lower-privilege `xip` registers if external interrupts have been delegated to the lower-privilege modes.
RVIC.1.15	1.1 (p.5)	R	Each processor core must define a policy on how simultaneous active interrupts are taken by multiple hart contexts on the core.
RVIC.1.16	1.1 (p.5)	I	For the simple case of a single stack of hart contexts, one for each supported privileged mode, interrupts for higher-privilege contexts can preempt execution of interrupt handlers for lower-privilege contexts.
RVIC.1.17	1.1 (p.5)	O	A multithreaded processor core could run multiple independent interrupt handlers on different hart contexts at the same time.

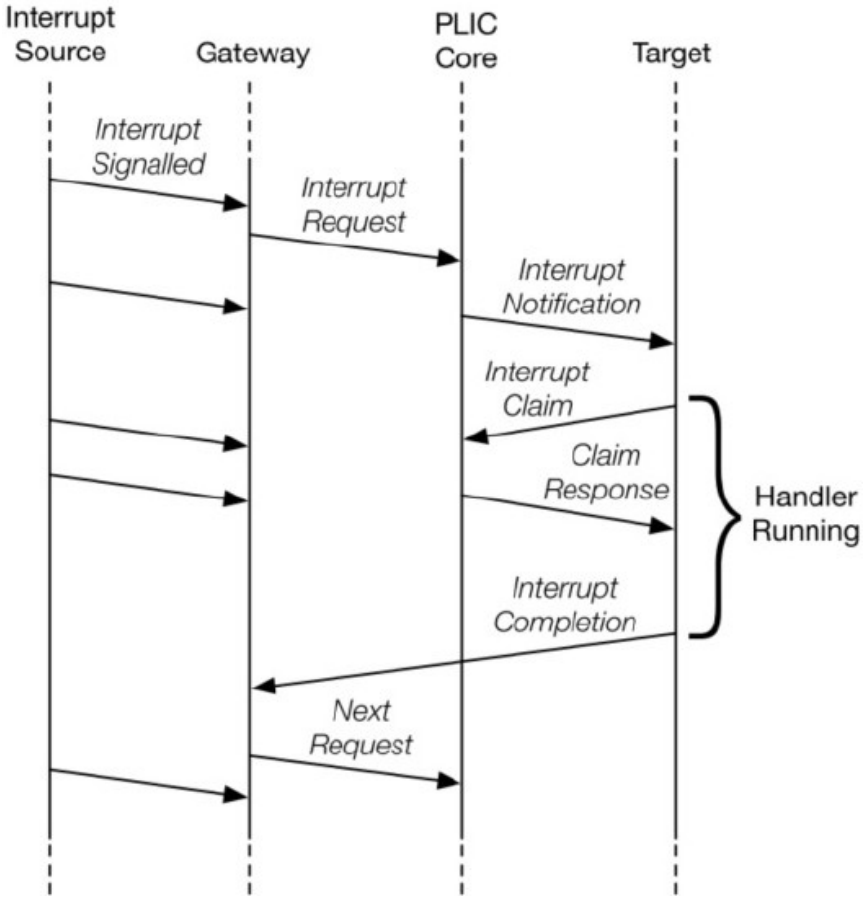
ID	REFERENCE	TYPE	DEFINITION
RVIC.1.18	1.1 (p.5)	O	A processor core could also provide hart contexts that are only used for interrupt handling to reduce interrupt service latency, and these might preempt interrupt handlers for other harts on the same core.
RVIC.1.19	1.1 (p.5)	R	The PLIC treats each interrupt target independently and does not take into account any interrupt prioritization scheme used by a component that contains multiple interrupt targets.
RVIC.1.20	1.1 (p.5)	R	As a result, the PLIC provides no concept of interrupt preemption or nesting so this must be handled by the cores hosting multiple interrupt target contexts.
RVIC.1.21	1.1 (p.6) Figure 1	R	RISC-V PLIC Interrupt Architecture Block Diagram



RVIC.1.22	1.2 (p.6)	H	Interrupt Gateways
RVIC.1.23	1.2 (p.6)	R	The interrupt gateways are responsible for converting global interrupt signals into a common interrupt request format, and for controlling the flow of interrupt requests to the PLIC core.
RVIC.1.24	1.2 (p.6)	R	At most one interrupt request per interrupt source can be pending in the PLIC core at any time, indicated by setting the source's IP bit.
RVIC.1.25	1.2 (p.6)	R	The gateway only forwards a new interrupt request to the PLIC core after receiving notification that the interrupt handler servicing the previous interrupt request from the same source has completed.
RVIC.1.26	1.2 (p.6)	R	If the global interrupt source uses level-sensitive interrupts, the gateway will convert the first assertion of the interrupt level into an interrupt request, but thereafter the gateway will not forward an additional interrupt request until it receives an interrupt completion message.

ID	REFERENCE	TYPE	DEFINITION
RVIC.1.27	1.2 (p.6)	R	On receiving an interrupt completion message, if the interrupt is level-triggered and the interrupt is still asserted, a new interrupt request will be forwarded to the PLIC core.
RVIC.1.28	1.2 (p.6)	R	The gateway does not have the facility to retract an interrupt request once forwarded to the PLIC core.
RVIC.1.29	1.2 (p.6)	R	If a level-sensitive interrupt source deasserts the interrupt after the PLIC core accepts the request and before the interrupt is serviced, the interrupt request remains present in the IP bit of the PLIC core and will be serviced by a handler, which will then have to determine that the interrupt device no longer requires service.
RVIC.1.30	1.2 (p.6)	R	If the global interrupt source was edge-triggered, the gateway will convert the first matching signal edge into an interrupt request.
RVIC.1.31	1.2 (p.6)	O	Depending on the design of the device and the interrupt handler, in between sending an interrupt request and receiving notice of its handler's completion, the gateway might either ignore additional matching edges or increment a counter of pending interrupts.
RVIC.1.32	1.2 (p.6, p.7)	R	In either case, the next interrupt request will not be forwarded to the PLIC core until the previous completion message has been received.
RVIC.1.33	1.2 (p.7)	R	If the gateway has a pending interrupt counter, the counter will be decremented when the interrupt request is accepted by the PLIC core.
RVIC.1.34	1.2 (p.7)	R	Unlike dedicated-wire interrupt signals, message-signalled interrupts (MSIs) are sent over the system interconnect via a message packet that describes which interrupt is being asserted.
RVIC.1.35	1.2 (p.7)	R	The message is decoded to select an interrupt gateway, and the relevant gateway then handles the MSI similar to an edge-triggered interrupt.
RVIC.1.36	1.3 (p.7)	H	Interrupt Notifications
RVIC.1.37	1.3 (p.7)	R	Each interrupt target has an external interrupt pending (EIP) bit in the PLIC core that indicates that the corresponding target has a pending interrupt waiting for service.
RVIC.1.38	1.3 (p.7)	R	The value in EIP can change as a result of changes to state in the PLIC core, brought on by interrupt sources, interrupt targets, or other agents manipulating register values in the PLIC.
RVIC.1.39	1.3 (p.7)	R	The value in EIP is communicated to the destination target as an interrupt notification.
RVIC.1.40	1.3 (p.7)	R	If the target is a RISC-V hart context, the interrupt notifications arrive on the `meip`/`seip` bits depending on the privilege level of the hart context.
RVIC.1.41	1.3 (p.7)	C	(In simple systems, the interrupt notifications will be simple wires connected to the processor implementing a hart. In more complex platforms, the notifications might be routed as messages across a system interconnect.)
RVIC.1.42	1.3 (p.7)	R	The PLIC hardware only supports multicasting of interrupts, such that all enabled targets will receive interrupt notifications for a given active interrupt.
RVIC.1.43	1.3 (p.7)	C	(Multicasting provides rapid response since the fastest responder claims the interrupt, but can be wasteful in high-interrupt-rate scenarios if multiple harts take a trap for an interrupt that only one can successfully claim. Software can modulate the PLIC IE bits as part of each interrupt handler to provide alternate policies, such as interrupt affinity or round-robin unicasting.)

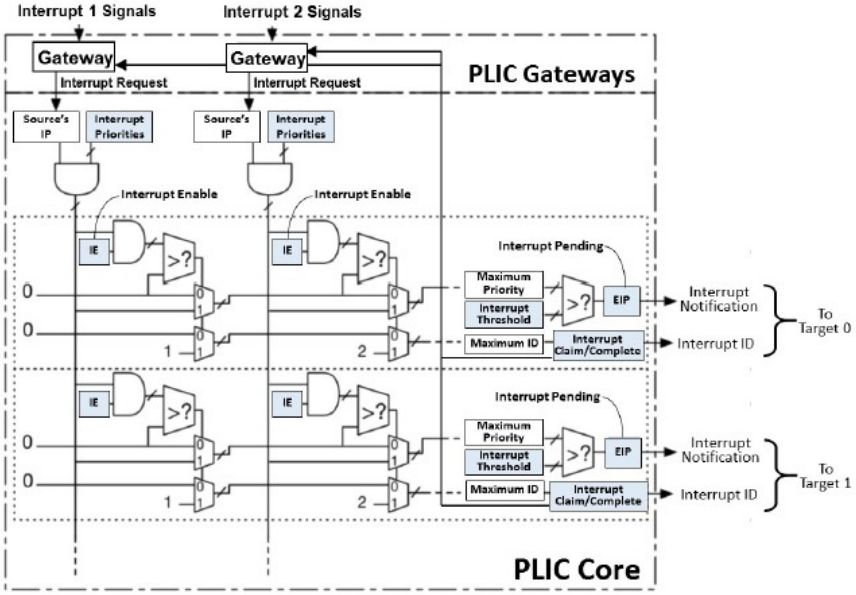
ID	REFERENCE	TYPE	DEFINITION
RVIC.1.44	1.3 (p.7)	I	Depending on the platform architecture and the method used to transport interrupt notifications, these might take some time to be received at the targets.
RVIC.1.45	1.3 (p.7)	R	The PLIC is guaranteed to eventually deliver all state changes in EIP to all targets, provided there is no intervening activity in the PLIC core.
RVIC.1.46	1.3 (p.7)	C	(The value in an interrupt notification is only guaranteed to hold an EIP value that was valid at some point in the past. In particular, a second target can respond and claim an interrupt while a notification to the first target is still in flight, such that when the first target tries to claim the interrupt it finds it has no active interrupts in the PLIC core.)
RVIC.1.47	1.4 (p.7)	H	Interrupt Identifiers (IDs)
RVIC.1.48	1.4 (p.7)	R	Global interrupt sources are assigned small unsigned integer identifiers, beginning at the value 1.
RVIC.1.49	1.4 (p.7)	R	An interrupt ID of 0 is reserved to mean “no interrupt”.
RVIC.1.50	1.4 (p.7)	R	Interrupt identifiers are also used to break ties when two or more interrupt sources have the same assigned priority.
RVIC.1.51	1.4 (p.7)	R	Smaller values of interrupt ID take precedence over larger values of interrupt ID.
RVIC.1.52	1.5 (p.7)	H	Interrupt Flow
RVIC.1.53	1.5 (p.7) Figure 2	R	Below figure shows the messages flowing between agents when handling interrupts via the PLIC. PLIC Interrupt Flow.



ID	REFERENCE	TYPE	DEFINITION
RVIC.1.54	1.5 (p.7, p.8)	R	• Global interrupts are sent from their source to an interrupt gateway that processes the interrupt signal from each source • Global interrupts are sent from their source to an interrupt gateway that processes the interrupt signal from each source • Interrupt gateway then sends a single interrupt request to the PLIC core, which latches these in the core interrupt pending bits (IP). • The PLIC core forwards an interrupt notification to one or more targets if the targets have any pending interrupts enabled, and the priority of the pending interrupts exceeds a per-target threshold. • When the target takes the external interrupt, it sends an interrupt claim request to retrieve the identifier of the highest priority global interrupt source pending for that target from the PLIC core. • PLIC core then clears the corresponding interrupt source pending bit. • After the target has serviced the interrupt, it sends the associated interrupt gateway an interrupt completion message • The interrupt gateway can now forward another interrupt request for the same source to the PLIC.

CHAPTER 2 RISC-V PLIC Operation Parameters

ID	REFERENCE	TYPE	DEFINITION
RVIC.2.1	2.0 (p.9)	H	RISC-V PLIC Operation Parameters
RVIC.2.2	2.0 (p.9)	I	General PLIC operation parameter register blocks are defined in this spec, those are: • **Interrupt Priorities registers:** The interrupt priority for each interrupt source. • **Interrupt Pending Bits registers:** The interrupt pending status of each interrupt source. • **Interrupt Enables registers:** The enablement of interrupt source of each context. • **Priority Thresholds registers:** The interrupt priority threshold of each context. • **Interrupt Claim registers:** The register to acquire interrupt source ID of each context. • **Interrupt Completion registers:** The register to send interrupt completion message to the associated gateway.
RVIC.2.3	2.0 (p.9) Figure 3	I	Below is the figure of PLIC Operation Parameter Block Diagram,



CHAPTER 3 Memory Map

ID	REFERENCE	TYPE	DEFINITION
RVIC.3.1	3.0 (p.10)	H	Memory Map
RVIC.3.2	3.0 (p.10)	I	The **base address of PLIC Memory Map** is platform implementation-specific.
RVIC.3.3	3.0 (p.10)	R	The memory-mapped registers specified in this chapter have a width of 32-bits.
RVIC.3.4	3.0 (p.10)	I	The bits are accessed atomically with LW and SW instructions.
RVIC.3.5	3.0 (p.10)	R	PLIC Memory Map base + `0x000000`: Reserved (interrupt source 0 does not exist) base + `0x000004`: Interrupt source 1 priority base + `0x000008`: Interrupt source 2 priority ... base + `0x000FFC`: Interrupt source 1023 priority base + `0x001000`: Interrupt Pending bit 0-31 base + `0x00107C`: Interrupt Pending bit 992-1023 ... base + `0x002000`: Enable bits for sources 0-31 on context 0 base + `0x002004`: Enable bits for sources 32-63 on context 0 ... base + `0x00207C`: Enable bits for sources 992-1023 on context 0 base + `0x002080`: Enable bits for sources 0-31 on context 1 base + `0x002084`: Enable bits for sources 32-63 on context 1 ... base + `0x0020FC`: Enable bits for sources 992-1023 on context 1 base + `0x002100`: Enable bits for sources 0-31 on context 2 base + `0x002104`: Enable bits for sources 32-63 on context 2 ... base + `0x00217C`: Enable bits for sources 992-1023 on context 2 ... base + `0x1F1F80`: Enable bits for sources 0-31 on context 15871 base + `0x1F1F84`: Enable bits for sources 32-63 on context 15871 base + `0x1F1FFC`: Enable bits for sources 992-1023 on context 15871 ... base + `0x1FFFFC`: Reserved base + `0x200000`: Priority threshold for context 0 base + `0x200004`: Claim/complete for context 0 base + `0x200008`: Reserved ... base + `0x200FFC`: Reserved base + `0x201000`: Priority threshold for context 1 base + `0x201004`: Claim/complete for context 1 ... base + `0x3FFF000`: Priority threshold for context 15871 base + `0x3FFF004`: Claim/complete for context 15871 base + `0x3FFF008`: Reserved ... base + `0x3FFFFFC`: Reserved
RVIC.3.6	3.1 (p.10)	I	Sections below (Chapter 4 - 9) describe the control register blocks of PLIC operation parameters.

CHAPTER 4 Interrupt Priorities

ID	REFERENCE	TYPE	DEFINITION
RVIC.4.1	4.0 (p.11)	H	Interrupt Priorities
RVIC.4.2	4.0 (p.11)	I	Interrupt priorities are small unsigned integers, with a platform-specific maximum number of supported levels.
RVIC.4.3	4.0 (p.11)	R	The priority value 0 is reserved to mean "never interrupt", and interrupt priority increases with increasing integer values.
RVIC.4.4	4.0 (p.11)	R	Each global interrupt source has an associated interrupt priority held in a memory-mapped register.
RVIC.4.5	4.0 (p.11)	O	Different interrupt sources need not support the same set of priority values.
RVIC.4.6	4.0 (p.11)	O	A valid implementation can hardwire all input priority levels.
RVIC.4.7	4.1 (p.11)	R	Interrupt source priority registers should be WARL fields to allow software to determine the number and position of read-write bits in each priority specification, if any.
RVIC.4.8	4.1 (p.11)	R	To simplify discovery of supported priority values, each priority register must support any combination of values in the bits that are variable within the register, i.e., if there are two variable bits in the register, all four combinations of values in those bits must operate as valid priority levels.
RVIC.4.9	4.1 (p.11)	R	If PLIC supports Interrupt Priorities, then each PLIC interrupt source can be assigned a priority by writing to its 32-bit memory-mapped priority register.
RVIC.4.10	4.1 (p.11)	R	A priority value of 0 is reserved to mean "never interrupt" and effectively disables the interrupt.
RVIC.4.11	4.1 (p.11)	R	Priority 1 is the lowest active priority while the maximum level of priority depends on PLIC implementation.
RVIC.4.12	4.1 (p.11)	R	Ties between global interrupts of the same priority are broken by the Interrupt ID; interrupts with the lowest ID have the highest effective priority.
RVIC.4.13	4.1 (p.11)	R	The base address of Interrupt Source Priority block within PLIC Memory Map region is fixed at `0x000000`.
RVIC.4.14	4.1 (p.11)	R	

PLIC Register Block Name	Function	Register Block Size in Byte	Description
Interrupt Source Priority	Interrupt Source Priority #0 to #1023	1024 * 4 = 4096(0x1000) bytes	This is a continuously memory block which contains PLIC Interrupt Source Priority. Total 1024 Interrupt Source Priority in this memory block. Interrupt Source Priority #0 is reserved which indicates it does not exist.

ID	REFERENCE	TYPE	DEFINITION
RVIC.4.15	4.2 (p.11)	R	PLIC Interrupt Source Priority Memory Map `0x000000`: Reserved (interrupt source 0 does not exist) `0x000004`: Interrupt source 1 priority `0x000008`: Interrupt source 2 priority ... `0x000FFC`: Interrupt source 1023 priority

CHAPTER 5 Interrupt Pending Bits

ID	REFERENCE	TYPE	DEFINITION
RVIC.5.1	5.0 (p.12)	H	Interrupt Pending Bits
RVIC.5.2	5.0 (p.12)	R	The current status of the interrupt source pending bits in the PLIC core can be read from the pending array, organized as 32-bit register.
RVIC.5.3	5.0 (p.12)	R	The pending bit for interrupt ID N is stored in bit (N mod 32) of word (N/32).
RVIC.5.4	5.0 (p.12)	R	Bit 0 of word 0, which represents the non-existent interrupt source 0, is hardwired to zero.
RVIC.5.5	5.0 (p.12)	R	A pending bit in the PLIC core can be cleared by setting the associated enable bit then performing a claim.
RVIC.5.6	5.0 (p.12)	R	The base address of Interrupt Pending Bits block within PLIC Memory Map region is fixed at `0x001000`.
RVIC.5.7	5.0 (p.12)	R	

PLIC Register Block Name	Function	Register Block Size in Byte	Description
Interrupt Pending Bits	Interrupt Pending Bit of Interrupt Source #0 to #N	1024 / 8 = 128(0x80) bytes	This is a continuously memory block contains PLIC Interrupt Pending Bits. Each Interrupt Pending Bit occupies 1-bit from this register block.

RVIC.5.8	5.1 (p.12)	R	PLIC Interrupt Pending Bits Memory Map `0x001000`: Interrupt Source #0 to #31 Pending Bits ... `0x00107C`: Interrupt Source #992 to #1023 Pending Bits

CHAPTER 6 Interrupt Enables

ID	REFERENCE	TYPE	DEFINITION
RVIC.6.1	6.0 (p.13)	H	Interrupt Enables
RVIC.6.2	6.0 (p.13)	R	Each global interrupt can be enabled by setting the corresponding bit in the **enables** register.
RVIC.6.3	6.0 (p.13)	R	The **enables** registers are accessed as a contiguous array of 32-bit registers, packed the same way as the **pending** bits.
RVIC.6.4	6.0 (p.13)	R	Bit 0 of enable register 0 represents the non-existent interrupt ID 0 and is hardwired to 0.
RVIC.6.5	6.0 (p.13)	R	PLIC has 15872 Interrupt Enable blocks for the contexts.
RVIC.6.6	6.0 (p.13)	I	How PLIC organizes interrupts for the contexts (Hart and privilege mode) is out of RISC-V PLIC specification scope, however it must be spec-out in vendor's PLIC specification.
RVIC.6.7	6.0 (p.13)	C	(A large number of potential IE bits might be hardwired to zero in cases where some interrupt sources can only be routed to a subset of targets. A larger number of bits might be wired to 1 for an embedded device with fixed interrupt routing. Interrupt priorities, thresholds, and hart-internal interrupt masking provide considerable flexibility in ignoring external interrupts even if a global interrupt source is always enabled.)
RVIC.6.8	6.0 (p.13)	R	The base address of Interrupt Enable Bits block within PLIC Memory Map region is fixed at `0x002000`.
RVIC.6.9	6.0 (p.13)	R	

PLIC Register Block Name	Function	Register Block Size in Byte	Description
Interrupt Enable Bits	Interrupt Enable Bit of Interrupt Source #0 to #1023 for 15872 contexts	(1024 / 8) * 15872 = 2031616 (0x1f0000) bytes	This is a continuously memory block contains PLIC Interrupt Enable Bits of 15872 contexts. Each Interrupt Enable Bit occupies 1-bit from this register block and total 15872 Interrupt Enable Bit blocks

ID	REFERENCE	TYPE	DEFINITION
RVIC.6.10			PLIC Interrupt Enable Bits Memory Map 0x002000: Interrupt Source #0 to #31 Enable Bits on context 0 ... 0x00207C: Interrupt Source #992 to #1023 Enable Bits on context 0 0x002080: Interrupt Source #0 to #31 Enable Bits on context 1 ... 0x0020FC: Interrupt Source #992 to #1023 Enable Bits on context 1 0x002100: Interrupt Source #0 to #31 Enable Bits on context 2 ... 0x00217C: Interrupt Source #992 to #1023 Enable Bits on context 2 0x002180: Interrupt Source #0 to #31 Enable Bits on context 3 ... 0x0021FC: Interrupt Source #992 to #1023 Enable Bits on context 3 ... ... 0x1F1F80: Interrupt Source #0 to #31 on context 15871 ... 0x1F1FFC: Interrupt Source #992 to #1023 on context 15871

CHAPTER 7 Priority Thresholds

ID	REFERENCE	TYPE	DEFINITION
RVIC.7.1	7.0 (p.14)	H	Priority Thresholds
RVIC.7.2	7.0 (p.14)	I	PLIC provides context based threshold register for the settings of a interrupt priority threshold of each context.
RVIC.7.3	7.0 (p.14)	R	The threshold register is a WARL field.
RVIC.7.4	7.0 (p.14)	R	The PLIC will mask all PLIC interrupts of a priority less than or equal to threshold. For example, a threshold value of zero permits all interrupts with non-zero priority.
RVIC.7.5	7.0 (p.14)	R	The base address of Priority Thresholds register block is located at 4K alignment starts from offset `0x200000`.
RVIC.7.6	7.0 (p.14)	R	

PLIC Register Block Name	Function	Register Block Size in Byte	Description
Priority Threshold	Priority Threshold for 15872 contexts	4096 * 15872 = 65011712 (0x3e00000) bytes	This is the register of Priority Thresholds setting for each context

RVIC.7.7	7.0 (p.14)	R	PLIC Interrupt Priority Thresholds Memory Map `0x200000`: Priority threshold for context 0 `0x201000`: Priority threshold for context 1 `0x202000`: Priority threshold for context 2 `0x203000`: Priority threshold for context 3 ... ... `0x3FFF000`: Priority threshold for context 15871

CHAPTER 8 Interrupt Claim Process

ID	REFERENCE	TYPE	DEFINITION
RVIC.8.1	8.0 (p.15)	H	Interrupt Claim Process
RVIC.8.2	8.0 (p.15)	I	Sometime after a target receives an interrupt notification, it might decide to service the interrupt.
RVIC.8.3	8.0 (p.15)	R	The target sends an interrupt claim message to the PLIC core, which will usually be implemented as a non-idempotent memory-mapped I/O control register read.
RVIC.8.4	8.0 (p.15)	R	On receiving a claim message, the PLIC core will atomically determine the ID of the highest-priority pending interrupt for the target and then clear down the corresponding source's IP bit.
RVIC.8.5	8.0 (p.15)	R	The PLIC core will then return the ID to the target.
RVIC.8.6	8.0 (p.15)	R	The PLIC core will return an ID of zero, if there were no pending interrupts for the target when the claim was serviced.
RVIC.8.7	8.0 (p.15)	R	After the highest-priority pending interrupt is claimed by a target and the corresponding IP bit is cleared, other lower-priority pending interrupts might then become visible to the target, and so the PLIC EIP bit might not be cleared after a claim.
RVIC.8.8	8.0 (p.15)	R	The interrupt handler can check the local `meip`/`seip`/`ueip` bits before exiting the handler, to allow more efficient service of other interrupts without first restoring the interrupted context and taking another interrupt trap.
RVIC.8.9	8.0 (p.15)	I	It is always legal for a hart to perform a claim even if the EIP is not set.
RVIC.8.10	8.0 (p.15)	I	In particular, a hart could set the threshold value to maximum to disable interrupt notifications and instead poll for active interrupts using periodic claim requests, though a simpler approach to implement polling would be to clear the external interrupt enable in the corresponding *x*`ie` register for privilege mode *x*.
RVIC.8.11	8.0 (p.15)	R	The PLIC can perform an interrupt claim by reading the claim/complete register, which returns the ID of the highest priority pending interrupt or zero if there is no pending interrupt.
RVIC.8.12	8.0 (p.15)	R	A successful claim will also atomically clear the corresponding pending bit on the interrupt source.
RVIC.8.13	8.0 (p.15)	R	The PLIC can perform a claim at any time and the claim operation is not affected by the setting of the priority threshold register.
RVIC.8.14	8.0 (p.15)	R	The Interrupt Claim Process register is context based and is located at (4K alignment + 4) starts from offset `0x200000`.
RVIC.8.15	8.0 (p.15)	R	

PLIC Register Block Name	Function	Register Block Size in Byte	Description
Interrupt Claim Register	Interrupt Claim Process for 15872 contexts	4096 * 15872 = 65011712 (0x3e00000) bytes	This is the register used to acquire interrupt ID for each context

ID	REFERENCE	TYPE	DEFINITION
RVIC.8.16			PLIC Interrupt Claim Process Memory Map `0x200004`: Interrupt Claim Process for context 0 `0x201004`: Interrupt Claim Process for context 1 `0x202004`: Interrupt Claim Process for context 2 `0x203004`: Interrupt Claim Process for context 3 ... ... `0x3FFF004`: Interrupt Claim Process for context 15871

CHAPTER 9 Interrupt Completion

ID	REFERENCE	TYPE	DEFINITION
RVIC.9.1	9.0 (p.16)	H	Interrupt Completion
RVIC.9.2	9.0 (p.16)	R	The PLIC signals it has completed executing an interrupt handler by writing the interrupt ID it received from the claim to the claim/complete register.
RVIC.9.3	9.0 (p.16)	R	The PLIC does not check whether the completion ID is the same as the last claim ID for that target.
RVIC.9.4	9.0 (p.16)	R	If the completion ID does not match an interrupt source that is currently enabled for the target, the completion is silently ignored.
RVIC.9.5	9.0 (p.16)	R	After a handler has completed service of an interrupt, the associated gateway must be sent an interrupt completion message, usually as a write to a non-idempotent memory-mapped I/O control register.
RVIC.9.6	9.0 (p.16)	R	The gateway will only forward additional interrupts to the PLIC core after receiving the completion message.
RVIC.9.7	9.0 (p.16)	R	The Interrupt Completion registers are context based and located at the same address with Interrupt Claim Process register, which is at (4K alignment + 4) starts from offset `0x200000`.
RVIC.9.8	9.0 (p.16)	R	**PLIC Register Block Name** | **Function** | **Register Block Size in Byte** | **Description** Interrupt Completion Register | Interrupt Completion for 15872 contexts | 4096 * 15872 = 65011712 (0x3e00000) bytes | This is register to write to complete Interrupt process
RVIC.9.9	9.0 (p.16)	R	PLIC Interrupt Completion Memory Map `0x200004`: Interrupt Completion for context 0 `0x201004`: Interrupt Completion for context 1 `0x202004`: Interrupt Completion for context 2 `0x203004`: Interrupt Completion for context 3 ... ... `0x3FFF004`: Interrupt Completion for context 15871