CVA6 – Platform Level Interrupt Controller (PLIC)

Design Document

Table of Contents

[Introduction 5](#_Toc88216142)

[Interrupts 5](#_Toc88216143)

[Hardware Interrupts 5](#_Toc88216144)

[Software Interrupts 5](#_Toc88216145)

[Interrupt Trigger 5](#_Toc88216146)

[Interrupt Sources 6](#_Toc88216147)

[Interrupt Handler / Interrupt Service Routine (ISR) 7](#_Toc88216148)

[PLIC Overview 8](#_Toc88216149)

[Interrupt Life Cycle 9](#_Toc88216150)

[PLIC High Level Design 10](#_Toc88216151)

[Interrupt Sources 10](#_Toc88216152)

[Interrupt Targets 10](#_Toc88216153)

[Interrupt Gateways 10](#_Toc88216154)

[Handling of Level Triggered Interrupts 11](#_Toc88216155)

[Handling of Edge Triggered Interrupts 11](#_Toc88216156)

[Handling of Message-Signaled Triggered Interrupts 11](#_Toc88216157)

[Interrupt Identifiers (IDs) 11](#_Toc88216158)

[Interrupt Priorities 11](#_Toc88216159)

[Interrupt Enables 12](#_Toc88216160)

[Enabling an interrupt in PLIC 13](#_Toc88216161)

[Disabling an interrupt in PLIC 13](#_Toc88216162)

[Interrupt Priority Thresholds 13](#_Toc88216163)

[Interrupt Notification 13](#_Toc88216164)

[Interrupt Claims 14](#_Toc88216165)

[Interrupt Completion 14](#_Toc88216166)

[Interrupt Flows 14](#_Toc88216167)

[CVA6 PLIC Memory Register Map 16](#_Toc88216168)

[PLIC RTL Design 17](#_Toc88216169)

[PLIC Top 17](#_Toc88216170)

[Functionality 17](#_Toc88216171)

[Module Diagram 18](#_Toc88216172)

[Signal Description 18](#_Toc88216173)

[PLIC Gateway 20](#_Toc88216174)

[Functionality 20](#_Toc88216175)

[Module Diagram 20](#_Toc88216176)

[Signal Description 20](#_Toc88216177)

[PLIC Registers 22](#_Toc88216178)

[Functionality 22](#_Toc88216179)

[Module Diagram 23](#_Toc88216180)

[Signal Description 23](#_Toc88216181)

[PLIC Target 25](#_Toc88216182)

[Functionality 25](#_Toc88216183)

[Module Diagram 26](#_Toc88216184)

[Signal Description 26](#_Toc88216185)

[Simulation Results 27](#_Toc88216186)

[Check 1: Interrupt enable and disable check 27](#_Toc88216187)

[Interrupt Enable Check 27](#_Toc88216188)

[Interrupt Disable Check 28](#_Toc88216189)

[Check 2: Interrupt priority set check 29](#_Toc88216190)

[Check 3: Threshold value set check 30](#_Toc88216191)

[Check 4: Interrupt claim check 31](#_Toc88216192)

[Check 5: Interrupt complete check 32](#_Toc88216193)

[Check 6: Multiple interrupt source interrupt detection and service checks 33](#_Toc88216194)

[Check 7: Level / Edge trigger interrupt detection check 34](#_Toc88216195)

List of Figures

[Figure 1: Platform Level Interrupt Controller (PLIC) conceptual block diagram 8](#_Toc88216196)

[Figure 2: High level interrupt flow chart PLIC to Processor 9](#_Toc88216197)

[Figure 3: PLIC Module block diagram 17](#_Toc88216198)

[Figure 4: plic\_top module 18](#_Toc88216199)

[Figure 5: rv\_plic\_gateway module 20](#_Toc88216200)

[Figure 6: Combinational logic for addressable register operations 22](#_Toc88216201)

[Figure 7: plic\_regmap module 23](#_Toc88216202)

[Figure 8: Working of PLIC target module 25](#_Toc88216203)

[Figure 9: rv\_plic\_target module 26](#_Toc88216204)

[Figure 10: Simulation - Interrupt Enable check 27](#_Toc88216205)

[Figure 11: Simulation - Interrupt disable check 28](#_Toc88216206)

[Figure 12: Simulation - Interrupt Priority Set Check 29](#_Toc88216207)

[Figure 13: Simulation - Threshold value set check 30](#_Toc88216208)

[Figure 14: Simulation - Interrupt Claim Check 31](#_Toc88216209)

[Figure 15: Simulation - Interrupt Complete Check 32](#_Toc88216210)

[Figure 16: Simulation - Multiple interrupt source interrupt detection and service check 33](#_Toc88216211)

[Figure 17: Simulation - Level / Edge trigger interrupt detection check 34](#_Toc88216212)

List of Tables

[Table 1: CVA6 PLIC Interrupt ID to interrupt source mapping 10](#_Toc88215394)

[Table 2: Valid priority value for CVA6 PLIC 12](#_Toc88215395)

[Table 3: PLIC Interrupt Priority Register Addresses 12](#_Toc88215396)

[Table 4: Interrupt Enable register write values for enabling interrupt sources 13](#_Toc88215397)

[Table 5: Flow for interrupt processing 15](#_Toc88215398)

[Table 6: CVA6 PLIC Register Map 16](#_Toc88215399)

[Table 7: Signal description for plic\_top module 19](#_Toc88215400)

[Table 8: Signal description for rv\_plic\_gateway module 21](#_Toc88215401)

[Table 9: Signal description for plic\_regmap module 24](#_Toc88215402)

[Table 10: Signal description for plic\_target moduel 26](#_Toc88215403)

# Introduction

This design document helps in understanding the working of CVA6 processor PLIC module and also covers the individual module functionality of PLIC RTL design. The document discusses the basics of interrupts, it handling and the need for PLIC for handling the interrupts.

## Interrupts

Interrupt is a signal emitted by hardware or software when a process needs immediate attention. It alerts the processor of a high priority process requiring interruption of the current working process.

### Hardware Interrupts

An electronic signal sent from an external device or hardware to communicate with the processor, indicating that it requires immediate attention. It can arrive asynchronously with respect to the processor clock, and at any time during instruction cycle. Hardware interrupts are classified into two types:

1. Maskable Interrupts

Processors have interrupt mask register that allows selective enabling and disabling of hardware interrupts. Every interrupt source has a bit dedicated to it in the mask register. If the bit is set, the interrupt is enabled and disabled when the bit is cleared. When the interrupt is disabled, the associated interrupt signal will be ignored by the processor. Signals which are affected by the mask are known as maskable interrupts.

1. Non-maskable Interrupts

The Non-maskable interrupts are highest priority activities that need to be processed immediately and under any situation. These signals are not affected by the interrupt mask and therefore cannot be disabled. For example, a timeout signal generated from a watchdog timer.

### Software Interrupts

The processor itself requests a software interrupt after execution of certain instructions or if particular conditions are satisfied. Every software interrupt signal is associated with a particular interrupt handler. A software interrupt may be intentionally caused by executing a special instruction which, by design, invokes an interrupt when executed. Such, instructions function similarly to subroutine calls and are used for a variety of purposes, such as requesting operating system services and interacting with device drivers.

Software interrupt may also be unexpectedly triggered by program execution errors. These interrupts typically are called traps or exceptions. For example, a divide-by-zero exception will be thrown if the processor executes a divide instruction with divisor equal to zero.

### Interrupt Trigger

Generally, the trigger signals are designed to trigger using either a logic signal level or a particular signal edge. These defines two types of interrupt triggering methods:

1. Level-triggered Interrupt

A level-triggered interrupt is requested by holding the interrupt signal at its particular (high or low) active logic level. A device invokes a level- triggered interrupt by driving the signal to and holding it at the active level. It negates the signal when the processor commands it to do so, typically after the device has been serviced.

The processor samples the interrupt input signal during each instruction cycle. The processor will recognize the interrupt request if the signal is asserted when the sampling occurs.

1. Edge-triggered Interrupt

An edge-triggered interrupt is an interrupt signaled by a level transition on the interrupt line, either a falling edge (high to low) or a rising edge (low to high). A device wishing to signal an interrupt drives a pulse onto the line and then releases the line to its inactive state. If the pulse is too short to be detected by polled I/O then special hardware is required to detect it.

### Interrupt Sources

RISC-V harts can have both local and global interrupt sources, out of which only global interrupt sources are handled by the PLIC.

#### Local Interrupt Sources

Each hart has a number of local interrupt sources that do not pass through the PLIC, including the standard software interrupts and timer interrupts for each privilege level. Local interrupts can be serviced quickly since there will be minimal latency between the source and the servicing hart. No arbitration is required to determine which hart will service the request.

#### Global Interrupt Sources

Global interrupt sources are those that are prioritized and distributed by the PLIC. Depending on the platform specific PLIC implementation, any global interrupt source could be routed to any hart context.

Global interrupt sources can take many forms, including level-triggered, edge-triggered, and message-signaled. Some sources might queue up a number of interrupt requests. All global interrupt sources are converted to a common interrupt request format for the PLIC.

## Interrupt Handler / Interrupt Service Routine (ISR)

The processor samples the interrupt trigger signal during each instruction cycle, and will respond to the trigger only if the signal is asserted when sampling occurs. The processor will begin interrupt processing at the next instruction boundary following a detected trigger, thus ensuring:

1. The Program Counter (PC) is saved in a known place.
2. All instructions before the one pointed to by the PC have fully executed.
3. No instruction beyond the one pointed to the PC has been executed, or any such instructions are undone before handling the interrupt.
4. The execution state of the instruction pointed to by the PC is known.

After ensuring the above mentioned conditions, the processor invokes the Interrupt Service Routine (ISR). Interrupt Service Routine is a special block of code associated with a specific interrupt condition. Interrupt handlers are initiated by hardware interrupts, software interrupt instructions, or software exceptions, and used for device drivers or transitions between protected modes of operations, such as system calls.

The traditional form of interrupt handler is the hardware interrupt handler. Hardware interrupts arise from electrical conditions or low-level protocols implemented in digital logic, are usually dispatched via a hard-coded table of interrupt vectors, asynchronously to the normal execution stream (as interrupt masking level permits), often using a separate stack, and automatically entering into a different execution context (privilege level) for the duration of the interrupt handler’s execution. In general, hardware interrupts and their handlers are used to handle high-priority conditions that require the interruption of the current code the processor is executing.

Later it was found convenient for software to be able to trigger the same mechanism by means of a software interrupt (a form of synchronous interrupt). Rather than using a hard-coded interrupt dispatch table at the hardware level, software interrupts are often implemented at the operating system level as a form of call back function.

## PLIC Overview

PLIC connects the global interrupt sources, which are usually I/O devices, to interrupt targets, which are usually hart contexts. The PLIC contains multiple interrupt gateways, one per interrupt source. Together with a PLIC core that performs interrupt prioritization and routing. Global interrupts are sent from their source to an interrupt gateway that processes the interrupt signal from each source and sends a single interrupt request to the PLIC core, which latches there in the core interrupt pending bits (IP).

Each interrupt source is assigned separate priority. The PLIC core contains a matrix of interrupt enable (IE) bits to select the interrupts that are enabled for each target. 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, which 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 and the interrupt gateway can now forward another interrupt request for the same source to the PLIC.

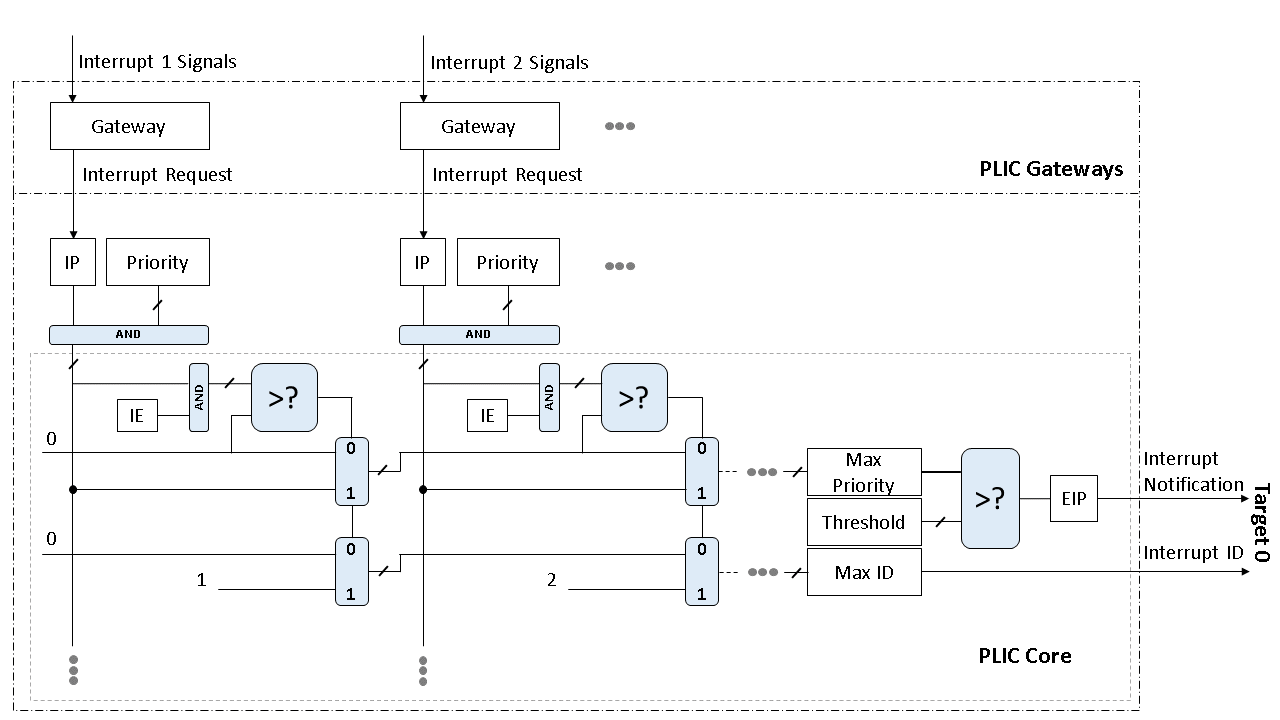


Figure : Platform Level Interrupt Controller (PLIC) conceptual block diagram

The above figure shows the first two of potentially many interrupt sources, and the first of potentially many interrupt targets. The figure is to show the logic of the PLIC operation and is not a realistic implementation strategy.

## Interrupt Life Cycle

Once the interrupt is triggered by the source, the gateway decodes the interrupt and raises an interrupt request with the PLIC core. The core then sets the Interrupt Pending bit for the particular source for which the interrupt was detected and assigns priority for the same.

After assigning priority, the PLIC core checks if the interrupt which was received is enabled. Further it checks whether the priority for the interrupt is greater than the threshold value set for the target. After confirming that the interrupt is for a given target, it raises the interrupt notification for the same.

At the processor side, after receiving the interrupt notification, the processor sends a read request to the PLIC. The PLIC then sends the interrupt ID of the detected interrupt. After checking if the received interrupt ID is valid, it services the interrupt and writes the same interrupt ID to the PLIC to indicate that the service for this interrupt is complete.

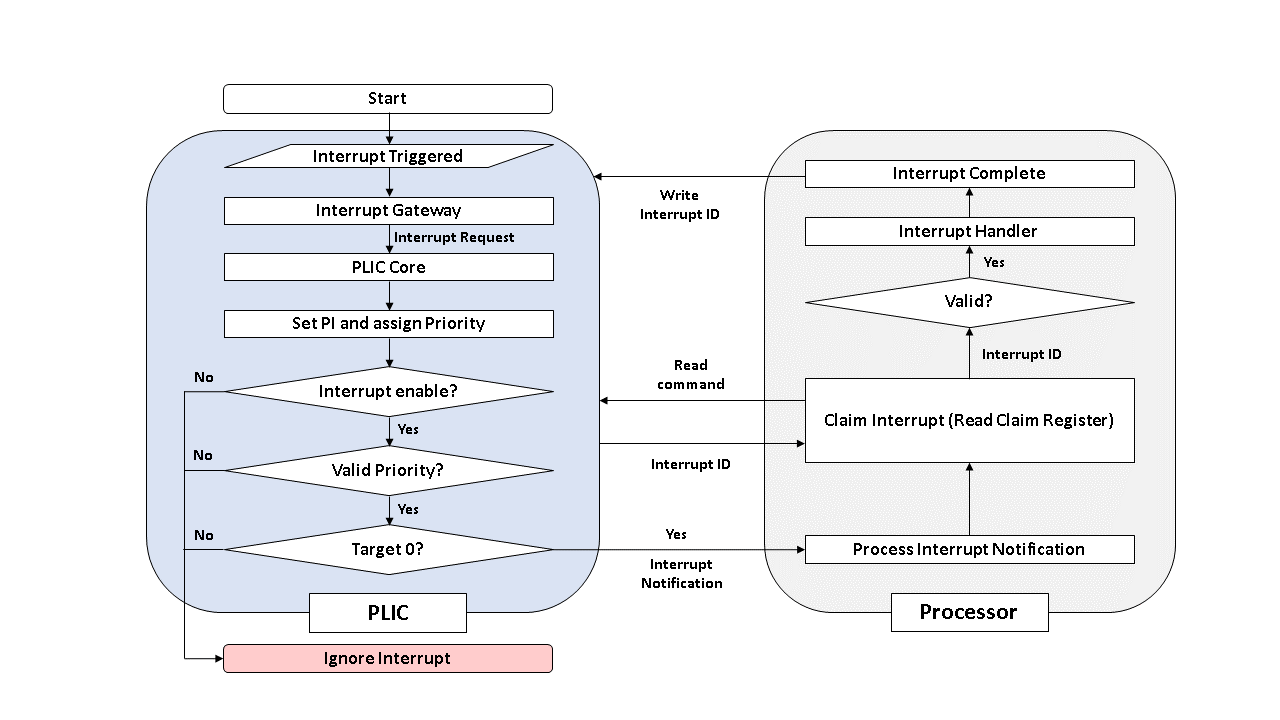


Figure : High level interrupt flow chart PLIC to Processor

# PLIC High Level Design

## Interrupt Sources

The source of interrupts for PLIC are devices connected to the SoC (GPIO, UART, I2C, etc…), these are global interrupt sources. Global interrupt sources can be level-triggered, edge triggered and message-signaled. In CVA6, all the interrupts are positive level triggered.

PLIC in CVA6 has 30 interrupt sources. 23 of these are exposed at the top level via the GPIO pins. Other interrupt sources are connected to UART, SPI, Ethernet and Timers. Global Interrupt ID 0 is reserved and hardwired to zero. Interrupt IDs starting from 1 are valid.

|  |  |
| --- | --- |
| **Interrupt ID** | **Source** |
| 1 | UART |
| 2 | SPI |
| 3 | Ethernet |
| 4 | Timer 0 (OVF) |
| 5 | Timer 0 (CMP) |
| 6 | Timer 1 (OVF) |
| 7 | Timer 1 (CMP) |
| 8 – 30 | *Reserved* |

Table : CVA6 PLIC Interrupt ID to interrupt source mapping

## Interrupt Targets

Interrupt targets are usually hart contexts, where a hart context is a given privilege mode on a given hart. There are other possible interrupt targets such as DMA engines. Each processor core defines a policy on how simultaneous interrupts are taken by multiple hart contexts on the core.

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. 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.

CVA6 PLIC can be configured for two targets.

## Interrupt Gateways

The interrupt gateways are responsible for converting global interrupt signals into common interrupt request format, and for controlling the flow of interrupt requests to the PLIC core. 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. 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.

### Handling of Level Triggered Interrupts

If the global interrupt source uses level-sensitive interrupts, the gateway will convert the first assertion of the interrupt level into interrupt request, but thereafter the gateway will not forward an additional interrupt request until it receives an interrupt completion message. On 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. 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.

### Handling of Edge Triggered Interrupts

If the global interrupt source is edge triggered, the gateway will convert the first matching signal edge into an interrupt request. 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. In either case, the next interrupt request will not be forwarded to the PLIC core until the previous completion message is received. If the gateway has pending interrupt counter, the counter will be decremented when the interrupt request is accepted by the PLIC core.

### Handling of Message-Signaled Triggered Interrupts

Message-signaled interrupts (MSIs) are sent over the system interconnect via a message packet that describes which interrupt is being asserted. The message is decoded to select an interrupt gateway, and the relevant gateway then handles the MSI similar to an edge triggered interrupt.

In CVA6, all the interrupts are positive level triggered.

## Interrupt Identifiers (IDs)

Global interrupt sources are small unsigned integer identifiers, beginning at the value 1. An interrupt ID of 0 is reserved to mean “no interrupt”.

Interrupt identifiers are also used to break ties when two or more interrupt sources have the same assigned priority. Smaller values of interrupt ID take precedence over larger values of interrupt ID.

CVA6 has 30 interrupt sources, thus the interrupt IDs vary from 1-30.

## Interrupt Priorities

Interrupt priorities are small unsigned integers, with a platform-specific maximum number of supported levels. The priority value of 0 is reserved to mean “never interrupt”, and the interrupt priority increases with increasing integer values.

Each global interrupt source has an associated interrupt priority held in a platform specific memory mapped register. Different interrupt sources need not support the same set of priority values. A valid implementation can hardwire all input priority levels. 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. If there are two variable bits in the register, all four combinations of values in those bits must operate as valid priority levels.

In degenerate cases, all priorities can be hardwired to the value 1, in which case input priorities are effectively determined by the interrupt ID.

CVA6 PLIC offers a maximum priority of 7. Thus, the priority of an interrupt can be set in 0 to 7.

|  |  |  |
| --- | --- | --- |
| **Priority level** | **Priority value** | **Hex value** |
| 1 | 3’b000 | 0x0 |
| 2 | 3’b001 | 0x1 |
| 3 | 3’b010 | 0x2 |
| 4 | 3’b011 | 0x3 |
| 5 | 3’b100 | 0x4 |
| 6 | 3’b101 | 0x5 |
| 7 | 3’b110 | 0x6 |
| 8 | 3’b111 | 0x7 |

Table : Valid priority value for CVA6 PLIC

|  |  |
| --- | --- |
| **Interrupt ID** | **Priority Register Values** |
| 0 | 0x0C000000 |
| 1 | 0x0C000004 |
| 2  : | 0X0C000008  : |
| 31 | 0X0C00007C |

Table : PLIC Interrupt Priority Register Addresses

## Interrupt Enables

Each target has a vector of interrupt enable (IE) bits, one per interrupt source. The target will not receive interrupts from sources that are disabled. The IE bits for a single target should be packed together as a bit vector in platform-specific memory-mapped control registers to support rapid context switching of the IE bits for the target. CVA6 PLIC has an internal 32-bit Interrupt Enable register. Bit-0 is reserved for interrupt source 0. Bits 1 to 30 are assigned to interrupt source 1 to 30 respectively.

The enabled bit for interrupt ID N is stored in N mod 8 bit in the N/8\_th byte.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Interrupt ID | Byte Offset | Bit Position | Hex-Value | Description (Enable-1, Disable-0) |
| 0 | 0 | 0 | 0x00000000 | Hardwired to zero |
| 1 | 0 | 1 | 0x00000002 | Global Interrupt Source 1 |
| 2  : | 0 | 2 | 0x00000004 | Global Interrupt Source 2 |
| 7 | 0 | 7 | 0x00000080 | Global Interrupt Source 7 |
| 8  : | 1 | 0 | 0x00000100 | Global Interrupt Source 8 |
| 16 | 2 | 0 | 0X00010000 | Global Interrupt Source 16 |
| 17  : | 2 | 1 | 0X00020000 | Global Interrupt Source 17 |
| 29 | 3 | 5 | 0X20000000 | Global Interrupt Source 29 |
| 30 | 3 | 6 | 0X40000000 | Global Interrupt Source 30 |

Table : Interrupt Enable register write values for enabling interrupt sources

### Enabling an interrupt in PLIC

* To enable an interrupt, the bit position corresponding to the interrupt source is set to 1 in Interrupt Enable register.
* Let *current value*, hold the value in the interrupt enable register.
* Let *new value* hold the value of interrupt enable register after *interrupt id* is reset.

### Disabling an interrupt in PLIC

* To disable an interrupt, the bit position corresponding to the interrupt source is set to 0 in Interrupt Enable register.
* Let *current value* hold the value in the interrupt enable register.
* Let *new value* hold the value of interrupt enable register after interrupt id is reset.

## Interrupt Priority Thresholds

Each interrupt target has an associated priority threshold, held in a platform-specific memory-mapped register. Only active interrupts that have a priority strictly greater than the threshold will cause an interrupt notification to be sent to the target. Different interrupt targets need not support the same set of priority threshold values. A threshold register should always be able to hold the value zero, in which case, no interrupts are masked. If implemented, the threshold register will usually also be able to hold the maximum priority level, in which case all the interrupts are masked.

The threshold value can be set in 0 to 7 for CVA6 PLIC.

## Interrupt Notification

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. The value of EIP can change as a result of changes to the state in the PLIC core, brought on by interrupt sources, interrupt targets, or other agents manipulating register values in the PLIC core. The value in EIP is communicated to the destination target as an interrupt notification.

In a simple system, the interrupt notifications will be simple wires connected to the processor implementing a hart. In more complex platforms, the notification might be routed as messages across a system interconnect.

The PLIC hardware only supports multicasting of interrupts, such that all enabled targets will receive interrupt notification for a given active interrupt.

Depending on the platform architecture and the method used to transport interrupt notifications, these might take some time to be received at the targets. 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.

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.

## Interrupt Claims

After a target receives an interrupt notification, it might decide to service the interrupt. The target sends an interrupt claim message to the PLIC core, which will usually be implemented as a non-idempotent memory-mapped IO control register read. 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. The PLIC core will then return the ID to the target. The PLIC core will return the ID of zero, if there were no pending interrupts for the target when the claim was serviced.

After the highest-priority pending interrupt is claimed by the 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.

It is always legal for a hart to perform a claim even if the EIP is not set. 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.

## Interrupt Completion

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. The gateway will only forward additional interrupts to the PLIC core after receiving the completion message.

## Interrupt Flows

The figure below shows messages flowing between the agents when handling interrupts via the PLIC. The gateway will only forward a single interrupt request at a time to the PLIC, and not forward subsequent interrupt requests until an interrupt completion is received. The PLIC will set the IP bit once it accepts an interrupt request from the gateway, and sometime later forward an interrupt notification to the target. The target might take a while to respond to a new interrupt arriving, but will then send an interrupt claim request to the PLIC core to obtain the interrupt ID. The PLIC core will atomically return the ID and clear the corresponding IP bit, after which no other target can claim the same interrupt request. Once the handler has processed the interrupt, it sends an interrupt completion message to the gateway to allow a new interrupt request.

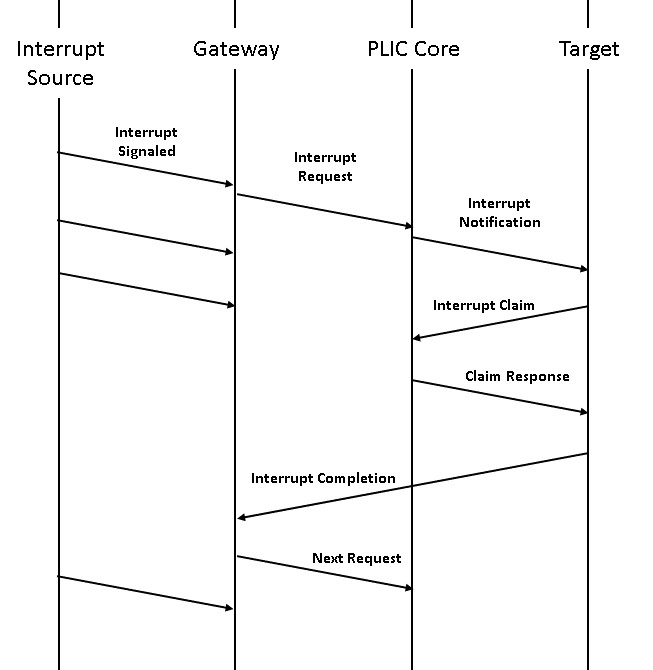


Table : Flow for interrupt processing

# CVA6 PLIC Memory Register Map

|  |  |  |
| --- | --- | --- |
| **Sr. No.** | **Register Address (Hex)** | **Register Description** |
| 1 | C000000 | Priority value (default 0) for Interrupt Source 0 (RESERVED) |
| 2 | C000004 | Priority value (default 0) for Interrupt Source 1 |
| 3 | C000008 | Priority value (default 0) for Interrupt Source 2 |
| 4 | C00000C | Priority value (default 0) for Interrupt Source 3 |
| 5 | C000010 | Priority value (default 0) for Interrupt Source 4 |
| 6 | C000014 | Priority value (default 0) for Interrupt Source 5 |
| 7 | C000018 | Priority value (default 0) for Interrupt Source 6 |
| 8 | C00001C | Priority value (default 0) for Interrupt Source 7 |
| 9 | C000020 | Priority value (default 0) for Interrupt Source 8 |
| 10 | C000024 | Priority value (default 0) for Interrupt Source 9 |
| 11 | C000028 | Priority value (default 0) for Interrupt Source 10 |
| 12 | C00002C | Priority value (default 0) for Interrupt Source 11 |
| 13 | C000030 | Priority value (default 0) for Interrupt Source 12 |
| 14 | C000034 | Priority value (default 0) for Interrupt Source 13 |
| 15 | C000038 | Priority value (default 0) for Interrupt Source 14 |
| 16 | C00003C | Priority value (default 0) for Interrupt Source 15 |
| 17 | C000040 | Priority value (default 0) for Interrupt Source 16 |
| 18 | C000044 | Priority value (default 0) for Interrupt Source 17 |
| 19 | C000048 | Priority value (default 0) for Interrupt Source 18 |
| 20 | C00004C | Priority value (default 0) for Interrupt Source 19 |
| 21 | C000050 | Priority value (default 0) for Interrupt Source 20 |
| 22 | C000054 | Priority value (default 0) for Interrupt Source 21 |
| 23 | C000058 | Priority value (default 0) for Interrupt Source 22 |
| 24 | C00005C | Priority value (default 0) for Interrupt Source 23 |
| 25 | C000060 | Priority value (default 0) for Interrupt Source 24 |
| 26 | C000064 | Priority value (default 0) for Interrupt Source 25 |
| 27 | C000068 | Priority value (default 0) for Interrupt Source 26 |
| 28 | C00006C | Priority value (default 0) for Interrupt Source 27 |
| 29 | C000070 | Priority value (default 0) for Interrupt Source 28 |
| 30 | C000074 | Priority value (default 0) for Interrupt Source 29 |
| 31 | C000078 | Priority value (default 0) for Interrupt Source 30 |
| 32 | C002000 | Interrupt Enable (default 0) Register for Target 0. (Bit 0 is RESERVED) |
| 33 | C002080 | Interrupt Enable (default 0) Register for Target 1. (Bit 0 is RESERVED) |
| 34 | C200000 | Threshold Value (default 0) for Target 0 |
| 35 | C201000 | Threshold Value (default 0) for Target 1 |
| 36 | C200004 | Claim / Complete Interrupt ID for Target 0 |
| 37 | C201004 | Claim / Complete Interrupt ID for Target 1 |

Table : CVA6 PLIC Register Map

# PLIC RTL Design

## PLIC Top

### Functionality

* The top module provides interface for the PLIC and integrates the gateway, register and target modules.
* This module is configurable and is by default configured to include 30 interrupt sources and 2 targets.
* Along with integrating the sub-modules, this module also implements logic for claim and complete onehot0 vector generation. This claim and complete vector are input to the gateway and perform the function of clearing the interrupt pending bit.
* The priority and interrupt enable signals are converted from the user format to internally utilized module format for use of the gateway and target modules.
* Input to the module consists of the interrupt sources and the signals for configuring interrupt source type (level / edge trigger).
* The outputs consist of interrupt notification signal for each target, along with the request and response buses.
* The code for this module can be found at: [plic\_top.sv](https://github.com/pulp-platform/rv_plic/blob/ebe3e9888920f1f757b75bab81dc80f804896f57/rtl/plic_top.sv)

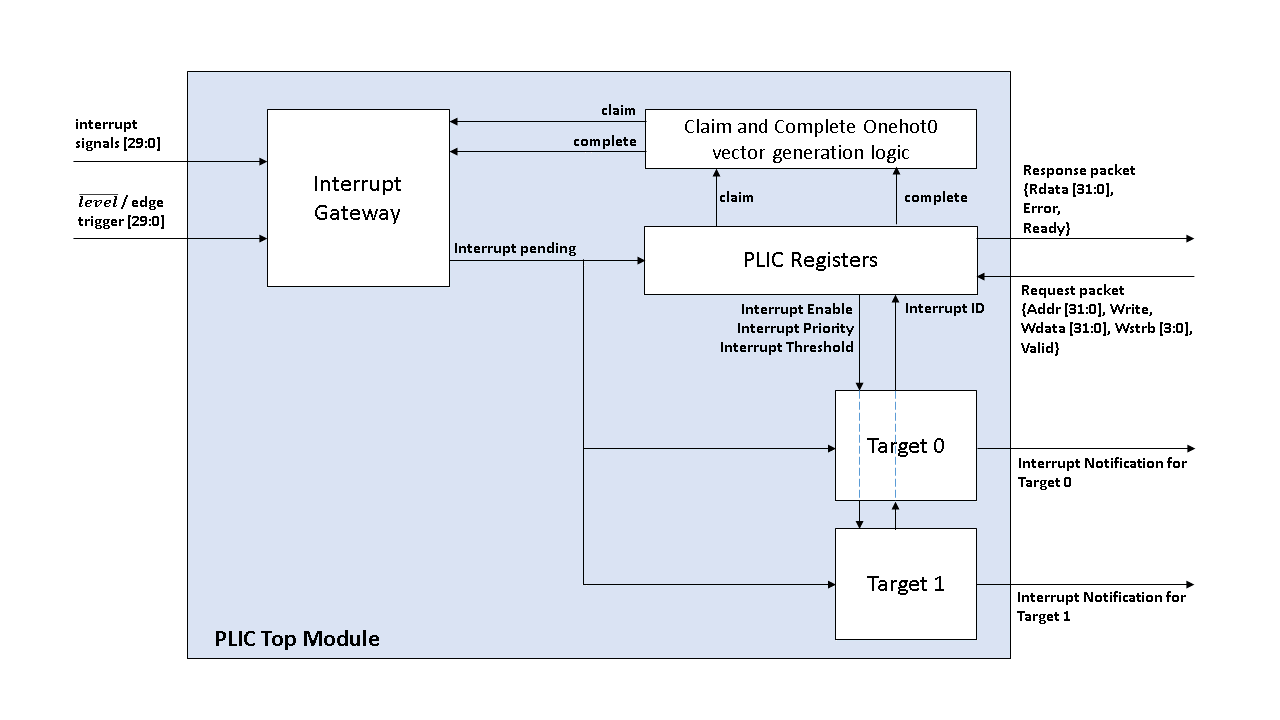


Figure : PLIC Module block diagram

### Module Diagram

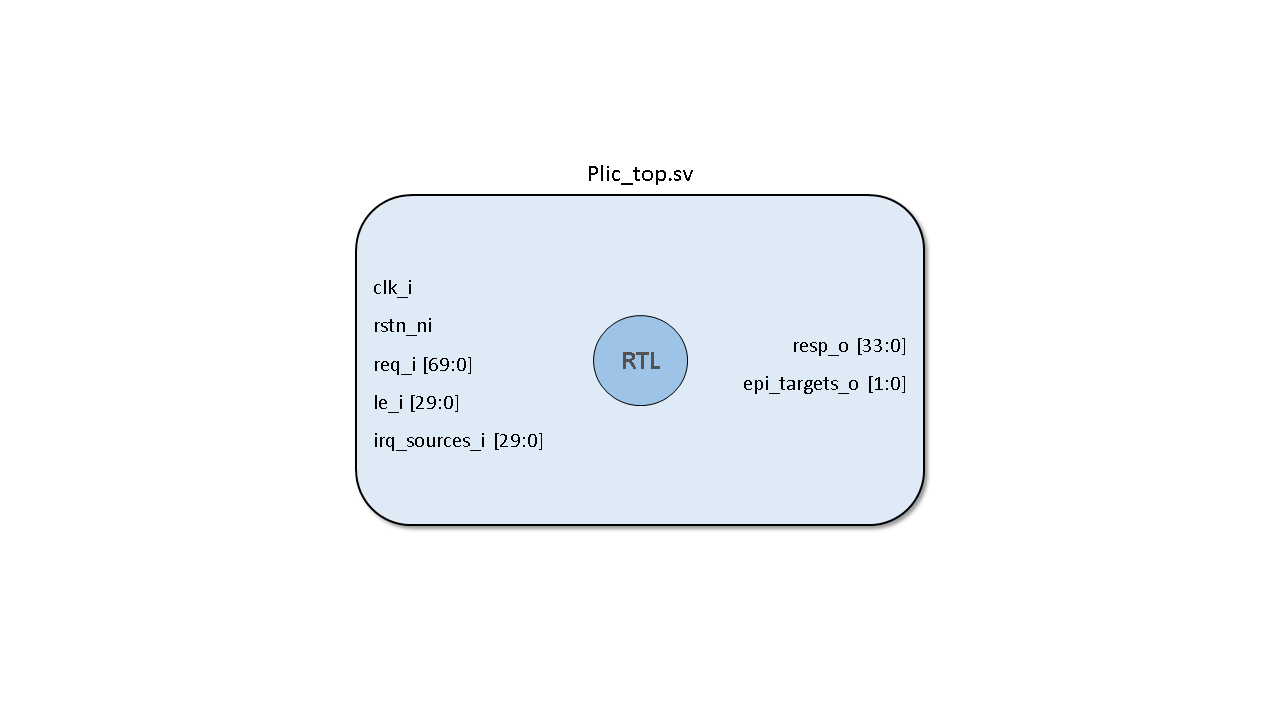


Figure : plic\_top module

### Signal Description

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Sr. No.** | **Name** | | **Input / Output/ Parameter** | **Bit Width / Type** | **Description** |
| 1 | N\_SOURCE | | Parameter | Int | Number of interrupt sources to be connected to the PLIC. |
| 2 | N\_TARGET | | Parameter | Int | Number of targets to be connected to the PLIC. |
| 3 | MAX\_PRIO | | Parameter | Int | Maximum priority value that can be set for an interrupt source. |
| 4 | SROW | | Parameter | Int | The upper limit of the interrupt ID value that can be assigned to an interrupt source. |
| 5 | clk\_i | | Input | 1 bit | Clock signal to the module. |
| 6 | rst\_ni | | input | 1 bit | Active low reset signal to the module. |
| 7 | req\_i  (This signal is a structure that defines the request channel for PLIC) | Addr | Input | 32 bit | This address bus is used to communicate to the PLIC register map for writing to the registers or reading from the registers. |
| Write | 1 bit | Asserting this bit will write to the address else perform a read from the address. |
| Wdata | 32 bit | When write bit is asserted, the data on these lines will be written to the addressed register state. |
| Wstrb | 4 bit | The write strobe allows to mask write data bytes when cleared. |
| Valid | 1 bit | Valid is asserted to indicate a valid request to the PLIC. |
| 8 | le\_i | | input | 30 bit | Each interrupt source has a bit dedicated for configuring the trigger type. A value of 1 on the line declares the interrupt source as edge trigger, else it is considered as level trigger. |
| 9 | irq\_sources\_i | | input | 30 bit | Interrupt sources incoming to the PLIC. |
| 10 | resp\_o  (This signal is a structure that define the response channel for the PLIC) | Rdata | Output | 32 bit | When valid signal is asserted and write bit is de-asserted, the PLIC will send the read data from the addressed register state on the response channel. |
| Error | 1 bit | This signal is asserted if any error occurs when request is raised to the PLIC. |
| Ready | 1 bit | This signal is asserted when the PLIC is ready to receive request. |
| 11 | epi\_targets\_o | | Output | 2 bit | Interrupt notification signals going to the target. |

Table : Signal description for plic\_top module

## PLIC Gateway

### Functionality

* The gateway module detects the level / edge triggered interrupt and raises an interrupt request on occurrence of the same.
* The PLIC gateway module keeps track of the interrupt pending bits in the PLIC. Assertion of interrupt pending bit indicates that an interrupt is detected for a particular source and the same needs to be serviced by the target.
* On detection of the interrupt request signal for a particular interrupt source, the module checks if the interrupt pending bit is already asserted. If it is asserted, the interrupt request is ignored, else the interrupt request is processed and the interrupt pending bit for the particular interrupt source is set.
* The interrupt pending bit for a particular interrupt source is de-asserted when the target claims the interrupt. Even though the interrupt pending bit is cleared, the module cannot generate an interrupt notification for the next interrupt or assert its interrupt pending bit until it receives an interrupt complete signal.
* Code for this module can be found at: [rv\_plic\_gateway.sv](https://github.com/pulp-platform/rv_plic/blob/ebe3e9888920f1f757b75bab81dc80f804896f57/rtl/rv_plic_gateway.sv)

### Module Diagram

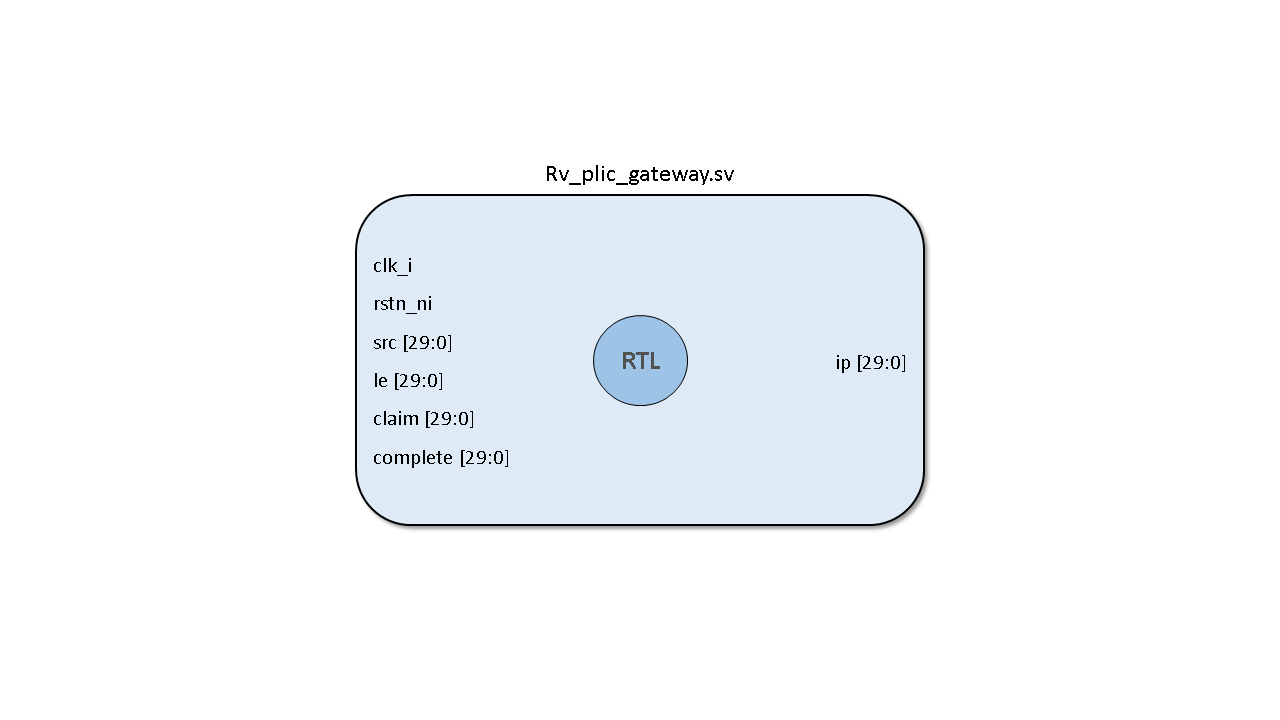


Figure : rv\_plic\_gateway module

### Signal Description

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No.** | **Name** | **Input / Output/ Parameter** | **Bit Width / Type** | **Description** |
| 1 | N\_SOURCE | Parameter | Int | Number of interrupt sources to be connected to the PLIC. |
| 2 | clk\_i | Input | 1 bit | Clock signal to the module. |
| 3 | rst\_ni | input | 1 bit | Active low reset signal to the module. |
| 4 | Src | Input | 30 bit | Interrupt sources incoming to the PLIC. |
| 5 | Le | Input | 30 bit | Level / Edge trigger configuration signal for interrupt sources. |
| 6 | Claim | Input | 30 bit | Onehot0 interrupt claim vector for clearing the interrupt pending bit |
| 7 | Complete | Input | 30 bit | Onehot0 interrupt complete vector for enabling successive interrupts from the serviced interrupt source. |
| 8 | ip | output | 30 bit | Interrupt pending bits, one dedicated to each interrupt source. |

Table : Signal description for rv\_plic\_gateway module

## PLIC Registers

### Functionality

* This module acts as a translation module and an interface between the internal signals of PLIC and the processor. It provides the processor a register map interface, which enables the processor to communicate with the PLIC in terms of read and write operations to the addressable locations.
* In this module, there are two combinational state machines, one is accessed when a read operation is performed and the other is accessed when a write operation is performed. Both the state machines contain conditional statements for each register address.
* When the request lines receive an address along with the read / write bit, the combinational logic first chooses the state machine to access based on the read / write bit value. Once the state machine is selected, the state which has the same value as the address line is selected and the operations in that state are performed.
* After completing the operation, the state machine goes to idle state where it waits for the next request to occur.
* Code for this module can be found at: [plic\_regmap.sv](https://github.com/pulp-platform/rv_plic/blob/ebe3e9888920f1f757b75bab81dc80f804896f57/rtl/plic_regmap.sv)

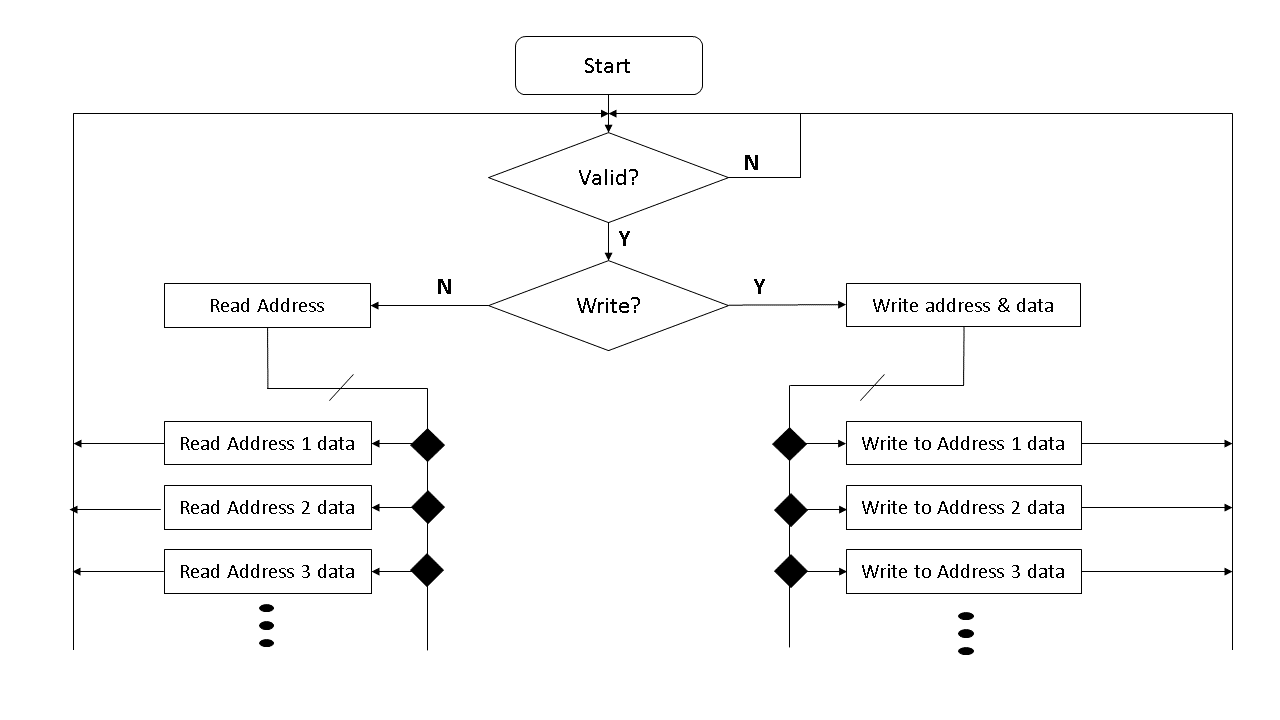


Figure : Combinational logic for addressable register operations

### Module Diagram

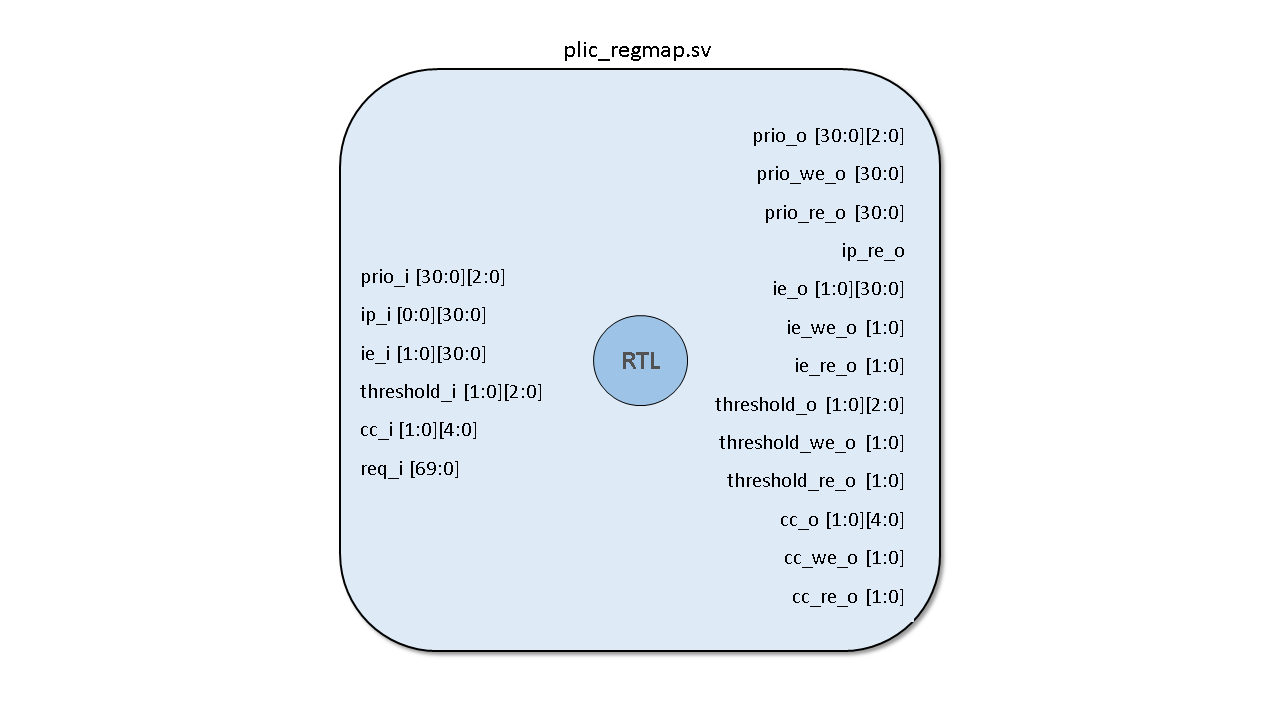


Figure : plic\_regmap module

### Signal Description

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Sr. No.** | **Name** | | **Input / Output/ Parameter** | **Bit Width / Type** | **Description** |
| 1 | Prio\_i | | input | 31 x 3 bits | Priority register value for each interrupt source. |
| 2 | Ip\_i | | input | 1 x 31 bit | Interrupt pending bits register value. |
| 3 | Ie\_i | | input | 2 x 31 bits | Interrupt Enable bits register value for both targets. |
| 4 | Threshold\_i | | input | 2 x 3 bits | Threshold register values that are set for both targets. |
| 5 | Cc\_i | | input | 2 x 5 | Interrupt ID of the interrupt with highest priority that needs to be serviced. |
| 6 | req\_i  (This signal is a structure that defines the request channel for PLIC) | Addr | Input | 32 bit | This address bus is used to communicate to the PLIC register map for writing to the registers or reading from the registers. |
| Write | 1 bit | Asserting this bit will write to the address else perform a read from the address. |
| Wdata | 32 bit | When write bit is asserted, the data on these lines will be written to the addressed register state. |
| Wstrb | 4 bit | The write strobe allows to mask write data bytes when cleared. |
| Valid | 1 bit | Valid is asserted to indicate a valid request to the PLIC. |
| 7 | Prio\_o | | output | 31 x 3 bits | Updated priority register value for each interrupt source to be written to internal registers. |
| 8 | Prio\_we\_o | | output | 31 bit | Priority write enable. When asserted, writes the updated interrupt priority value to the corresponding register. |
| 9 | Prio\_re\_o | | output | 31 bit | Priority register read enable (not used) |
| 10 | Ip\_re\_o | | output | 1 bit | Interrupt priority register read enable (not used). |
| 11 | Ie\_o | | output | 2 x 31 bits | Updated interrupt enable register value to be written to internal registers. |
| 12 | Ie\_we\_o | | output | 2 bits | Interrupt enable register write enable signal. When asserted, writes the updated interrupt enable value to the corresponding internal register. |
| 13 | Ie\_re\_o | | output | 2 bits | Interrupt enable register read enable (not used). |
| 14 | Threshold\_o | | output | 2 x 3 bits | Updated threshold value for the targets. |
| 15 | Threshold\_we\_o | | output | 2 bits | Threshold register write enable. When asserted, writes the updated threshold values to the corresponding registers. |
| 16 | Threshold\_re\_o | | output | 2 bits | Threshold register read enable (not used). |
| 17 | Cc\_o | | output | 2 x 5 bits | Updated value of the Interrupt ID for the interrupt for which the service has been completed. |
| 18 | Cc\_we\_o | | output | 2 bits | Complete Interrupt ID write enable register. When asserted, the updated value interrupt ID for which the service has been completed is updated to the corresponding internal register. |
| 19 | Cc\_re\_o | | output | 2 bits | Asserted when the processor wants to claim interrupt ID. Upon assertion, the internal interrupt ID value with highest priority is available for the processor to claim. |
| 20 | resp\_o  (This signal is a structure that define the response channel for the PLIC) | Rdata | Output | 32 bit | When valid signal is asserted and write bit is de-asserted, the PLIC will send the read data from the addressed register state on the response channel. |
| Error | 1 bit | This signal is asserted if any error occurs when request is raised to the PLIC. |
| Ready | 1 bit | This signal is asserted when the PLIC is ready to receive request. |

Table : Signal description for plic\_regmap module

## PLIC Target

### Functionality

* The PLIC target module keeps track of the enabled interrupts. Whenever an event occurs on the enabled interrupts, the interrupt pending bits are updated. The module keeps checking the interrupt pending bits for all the interrupt sources.
* For a particular source, if the interrupt is enabled and the interrupt pending bit is asserted, the module checks if the current priority value is greater than the maximum priority value. If true the maximum priority value is updated to the current priority value and the maximum interrupt ID value is also updated with the current interrupt ID value.
* If the updated maximum priority value is greater than the threshold value, interrupt notification is sent to the target.
* When the target sends a claim request, the updated maximum interrupt ID is sent to be serviced.
* Code for this module can be found at: [rv\_plic\_taget.sv](https://github.com/pulp-platform/rv_plic/blob/ebe3e9888920f1f757b75bab81dc80f804896f57/rtl/rv_plic_target.sv)

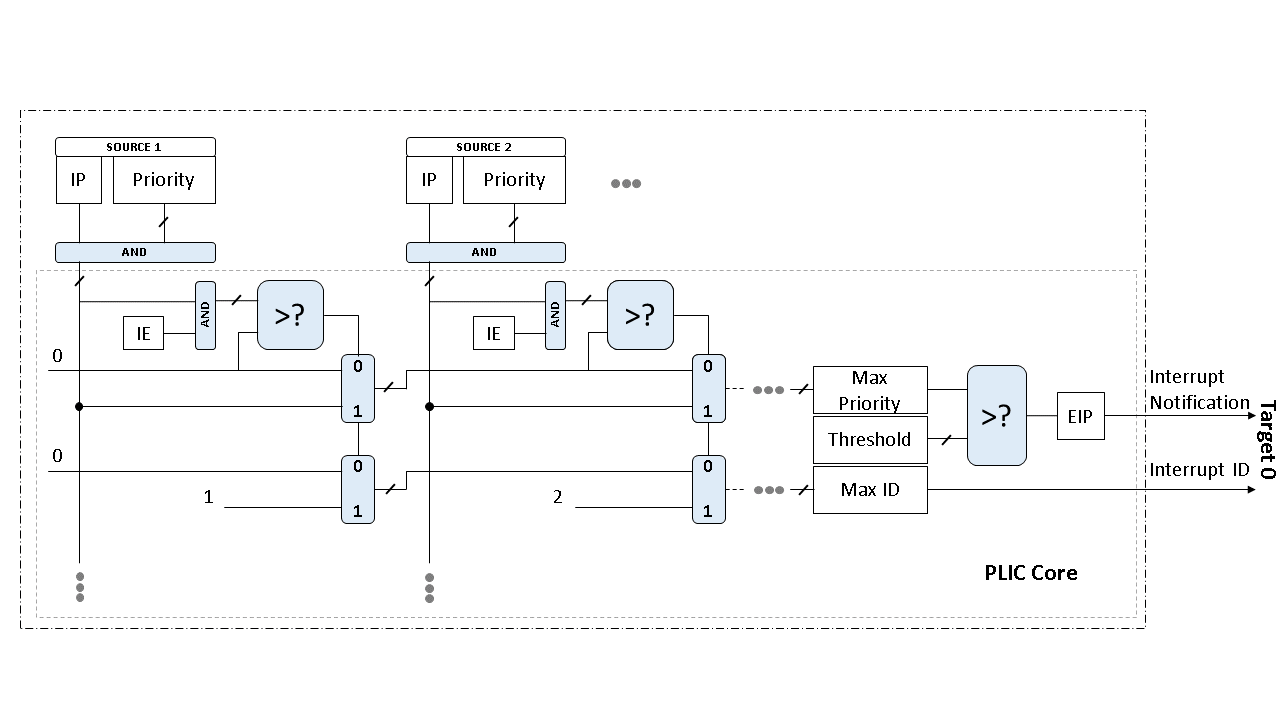


Figure : Working of PLIC target module

### Module Diagram

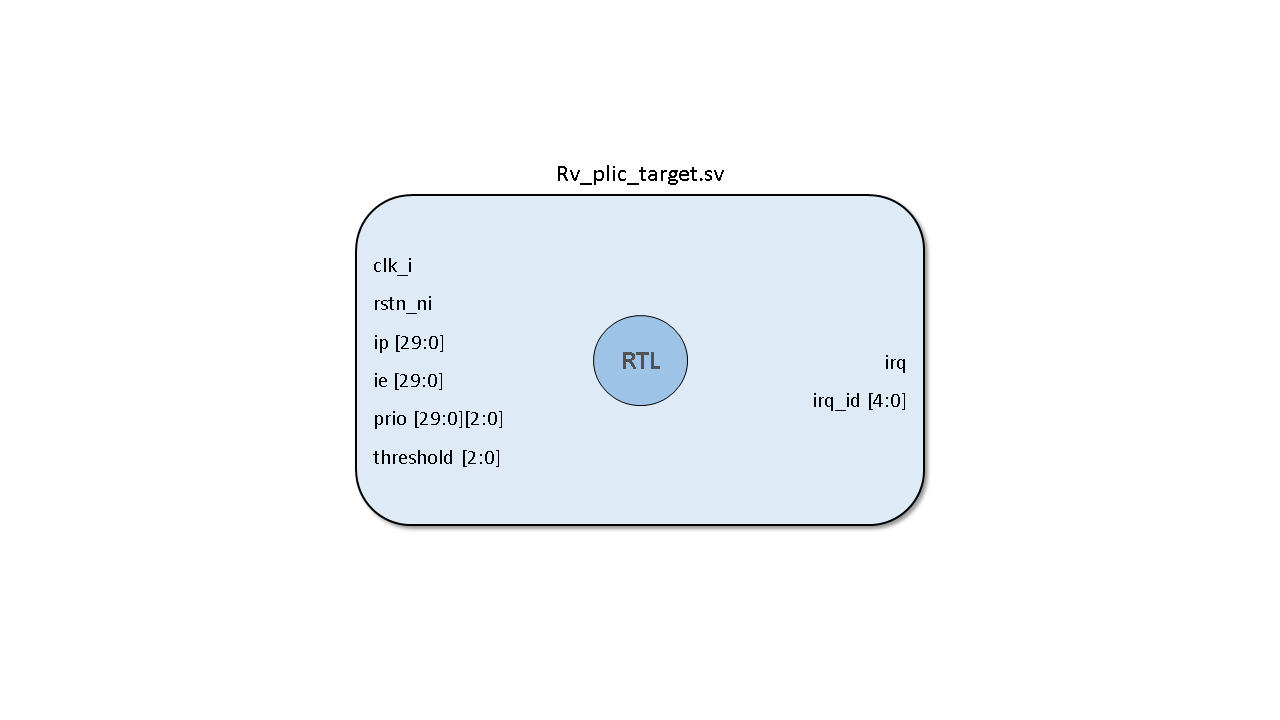


Figure : rv\_plic\_target module

### Signal Description

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No.** | **Name** | **Input / Output/ Parameter** | **Bit Width / Type** | **Description** |
| 1 | N\_SOURCE | Parameter | Int | Number of interrupt sources to be connected to the PLIC. |
| 2 | clk\_i | Input | 1 bit | Clock signal to the module. |
| 3 | rst\_ni | input | 1 bit | Active low reset signal to the module. |
| 4 | Src | Input | 30 bit | Interrupt sources incoming to the PLIC. |
| 5 | Le | Input | 30 bit | Level / Edge trigger configuration signal for interrupt sources. |
| 6 | Claim | Input | 30 bit | Onehot0 interrupt claim vector for clearing the interrupt pending bit |
| 7 | Complete | Input | 30 bit | Onehot0 interrupt complete vector for enabling successive interrupts from the serviced interrupt source. |
| 8 | ip | output | 30 bit | Interrupt pending bits, one dedicated to each interrupt source. |

Table : Signal description for rv\_plic\_target moduel

# Simulation Results

## Check 1: Interrupt enable and disable check

### Interrupt Enable Check

* In the interrupt enable check, the processor writes to the interrupt enable register to enable each interrupt source one by one, starting from interrupt source 1 up to interrupt source 30. Initially all the interrupt sources are disabled.
* To check if the interrupt enable commands are being processed by the PLIC, the internal interrupt enable register value is observed and it is ensured that all the bits of the 30 bit interrupt enable register are asserted.
* To enable the interrupt for Source N, set the Nth bit of the Interrupt Enable register (31 down to 0).
* Interrupt Enable Register Address for Target 0: 0x0C002000
* Interrupt Enable Register Address for Target 1: 0x0C002080
* Thus, to enable interrupt detection for Source N for Target 0, CVA6 should write value of 2^N to 0X0C002000 register.
* Bit-0 of Interrupt Enable register is RESERVED.

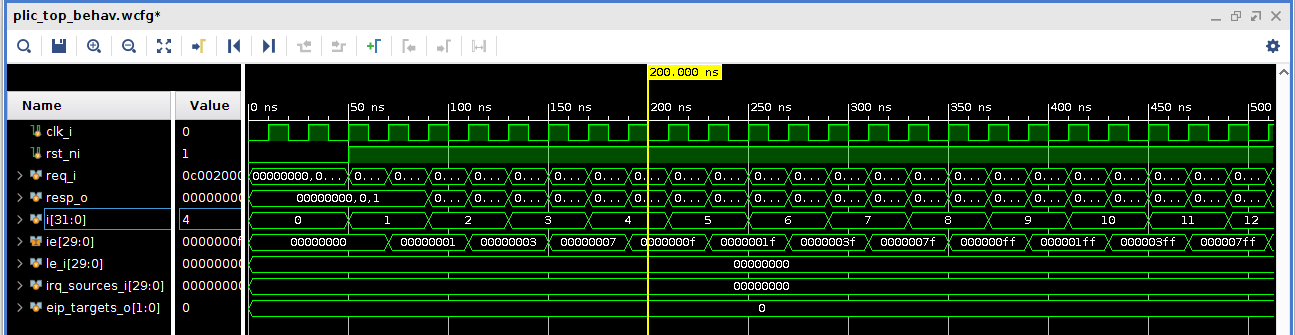


Figure : Simulation - Interrupt Enable check

* Operations performed for enabling an interrupt source:
  + Read from Interrupt Enable register (0X0C002000): The read operation is performed to know the initial status of the interrupt enable register. This retrieved value is or’ed with the value of the interrupt that is to be enabled and then written back to the register for enabling another interrupt without altering the enable status of other interrupts. As initially no interrupts are enabled, the read value returns 0.
  + Write to Interrupt Enable register (0X0C002000): As mentioned previously the or’ed value is written to the interrupt enable register to enable the required interrupt source. This or’ed value is sent as data.

### Interrupt Disable Check

* Similarly, for interrupt disable check, the processor writes to the interrupt enable register to disable each interrupt source one by one, starting from interrupt source 30 down to interrupt source 1. Initially all the interrupt sources are enabled.
* To check if the interrupt disable commands are being processed by the PLIC, the internal interrupt enable register value is observed and it is ensured that all the bits of the 30 bit interrupt enable register are cleared.
* To disable the interrupt for Source N, clear the Nth bit of the Interrupt Enable register (31 down to 0).

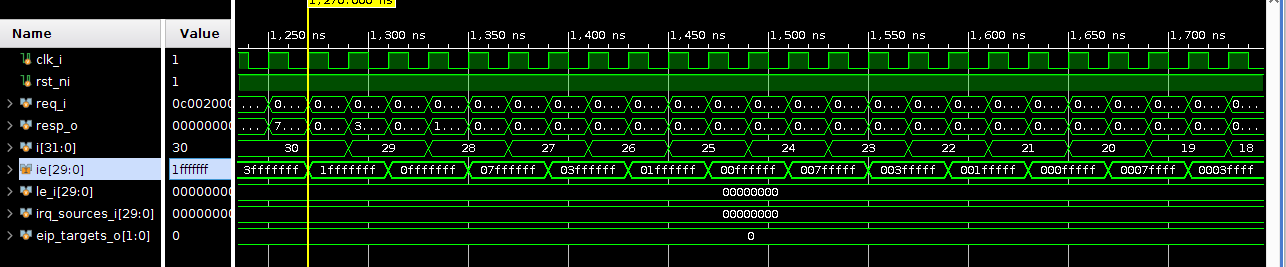


Figure : Simulation - Interrupt disable check

* Operations performed for disabling an interrupt source:
  + Read from Interrupt Enable register (0X0C002000): The read operation is performed to know the initial status of the interrupt enable register. This retrieved value is and’ed with the negated value of the interrupt that is to be disabled and then written back to the register for disabling another interrupt without altering the enable status of other interrupts.
  + Write to Interrupt Enable register (0X0C002000): As mentioned previously the and’ed value is written to the interrupt enable register to disable the required interrupt source. This and’ed value is sent as data.

## Check 2: Interrupt priority set check

* Each interrupt source has a priority value assigned to it. In this instance the priority can be assigned in 0 to 7.
* In the PLIC, there are dedicated registers assigned for holding the interrupt priority value for each interrupt source. The priority of the interrupt source can be changed by writing to these registers.
* Interrupt detection is carried out for sources which have a priority value greater than the threshold value set for a particular target.
* In this check we ensure that the priority value set commands are being processed by the PLIC by observing the internal register values of the interrupt priority registers for each interrupt source.



Figure : Simulation - Interrupt Priority Set Check

* Operations for setting interrupt source priority value:
  + Write to priority register: Write a value in 0 to 7 to the interrupt priority register for setting the interrupt priority.
  + Refer the *CVA6\_PLIC\_RegMap.xlxs* sheet for the interrupt priority register addresses.

## Check 3: Threshold value set check

* This check is performed to ensure that when the threshold value set command is sent to the PLIC, it processes the same and updates the threshold value for a target.
* To check this functionality the internal threshold register value is observed.
* Threshold register address for target 0: 0x0C200000
* Threshold register address for target 1: 0x0C201000

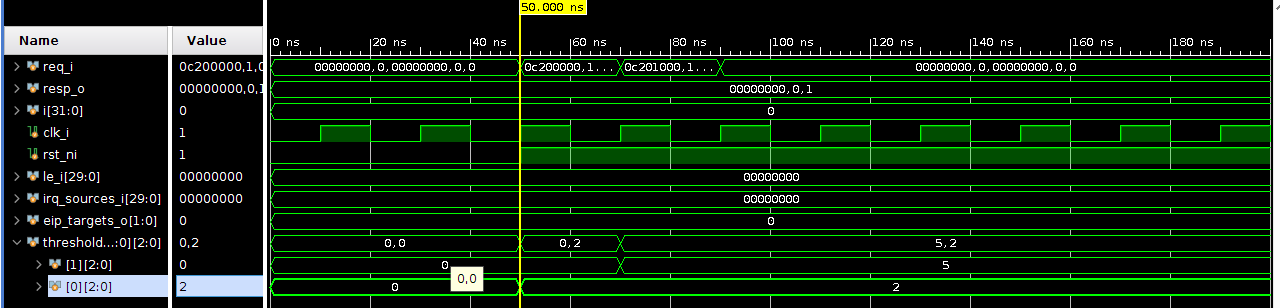


Figure : Simulation - Threshold value set check

* Steps to set the threshold register value:
  + Write to 0x0C200000 register address: The threshold value, which lies in 0 to 7 is written to the threshold register for target 0. Here, a threshold value of 2 is written to target 0 threshold.
  + Write to 0x0C201000 register address: The threshold value, which lies in 0 to 7 is written to the threshold register for target 1. Here, a threshold value of 5 is written to target 1 threshold.

## Check 4: Interrupt claim check

* Before performing the interrupt claim check, the interrupt is enabled and interrupt priority is set for the same.
* Interrupt is generated by asserting the interrupt source line for the enabled interrupt.
* This results in interrupt notification being generated by the PLIC, indicating that valid interrupt is detected.
* After receiving the interrupt notification, the processor sends a read request to the PLIC for claiming the interrupt. The PLIC then returns the interrupt id to be serviced to the processor and clears the interrupt pending bit against the interrupt id. This function is checked by observing the internal register for interrupt pending bits.

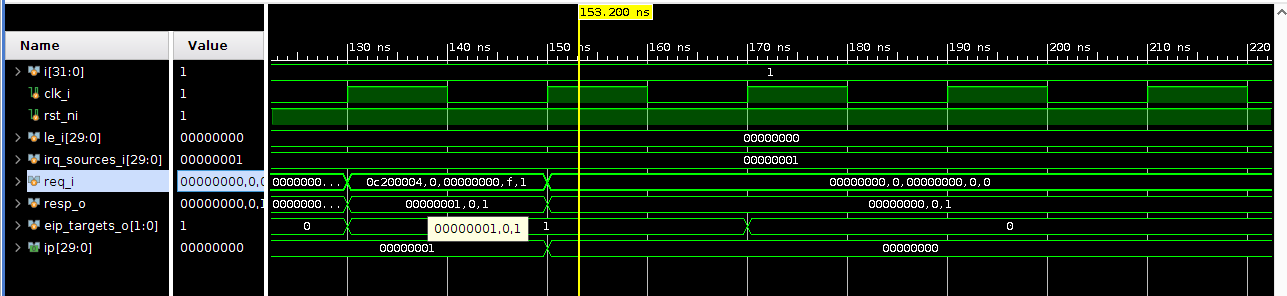


Figure : Simulation - Interrupt Claim Check

* Steps for interrupt claim check:
  + Enable (0X0C002000) the interrupt source (source 1) and set priority (0x0C000004). Generate an interrupt on the line.
  + Once detected send a read request on the claim interrupt register for target 0 (0x0C200004).
  + After receiving the claim request it can be observed that the interrupt pending bit for source 1 is cleared.
  + The PLIC will be able to accept the next interrupt from the same source and generate another interrupt notification after it receives the interrupt complete message from the processor.
* Interrupt Claim register address for target 0: 0x0C200004
* Interrupt Claim register address for target 1: 0x0C201004

## Check 5: Interrupt complete check

* Before performing the interrupt complete check, the interrupt is enabled and interrupt priority is set for the same. Interrupt is generated by asserting the interrupt source line for the enabled interrupt.
* Upon detection of interrupt notification, the interrupt is claimed. Thus, further check for interrupt complete can be performed.
* Upon claiming the interrupt, the PLIC clears the interrupt pending bit for the interrupt source, but it will still not process any further interrupt from the source or generate an interrupt notification for the processor until it receives an interrupt complete signal.
* Upon receiving the interrupt complete signal, the PLIC then accepts the next interrupt from the source and raises the interrupt notification for the same.
* Interrupt Complete register address for target 0: 0x0C200004. Interrupt Complete register address for target 1: 0x0C201004

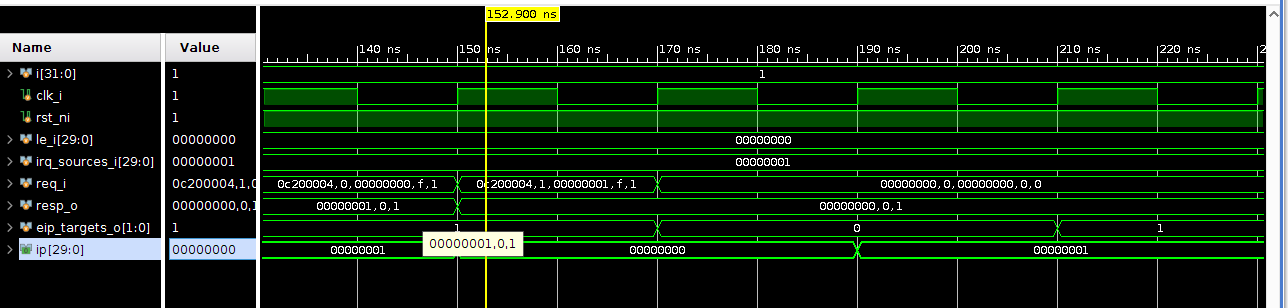


Figure : Simulation - Interrupt Complete Check

* Steps for interrupt completion:
  + Enable (0X0C002000) the interrupt source (source 1) and set priority (0x0C000004). Generate an interrupt on the line.
  + Once detected send a read request on the claim interrupt register for target 0 (0x0C200004).
  + For completing the interrupt, the interrupt id for which the service is completed is written to the interrupt claim register (0x0C200004, it is the same as interrupt claim register address).
  + It can be seen in the figure above that after the PLIC raises the interrupt notification signal again only after receiving the interrupt complete signal.

## Check 6: Multiple interrupt source interrupt detection and service checks

* In this check, multiple interrupt sources are enabled and assigned different priorities. The interrupts are generated for the same.
* This check observes if the interrupt claims are generated according to the assigned priorities.

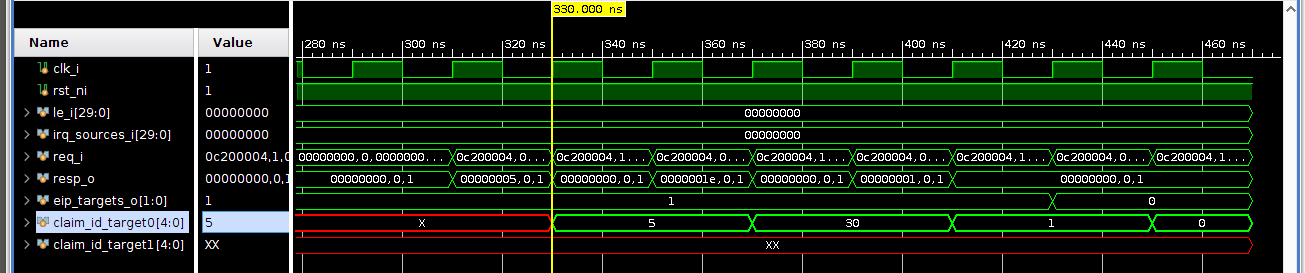


Figure : Simulation - Multiple interrupt source interrupt detection and service check

* Steps for multiple interrupt source interrupt detection and service checks:
  + Enable the interrupt sources 1, 5 and 30 and assign priorities 3, 7 and 4 respectively. Interrupt is generated on all enabled sources simultaneously.
  + According to the priorities assigned, upon claiming the interrupt the interrupt source claim order is 5,30 and 1 which can be observed in the figure above.

## Check 7: Level / Edge trigger interrupt detection check

* This check is performed to observe if the level / edge interrupt trigger detection functions properly.
* To begin the check, initially interrupt source 1 is enabled, its priority set and the level / edge bit is set to 0 indicating the trigger is level triggered.
* For level trigger check, an extended interrupt is generated. In this case, after first detection of interrupt, the PLIC generates interrupt notification and the processor then services the interrupt and then sends interrupt complete notification to the PLIC. Upon receiving this, since the interrupt trigger type is level interrupt, the interrupt is detected again and interrupt notification is raised. This process is repeated as long as the interrupt signal is asserted.
* Similarly, for edge trigger check, an extended interrupt is generated. In this case, the edge of the interrupt signal is detected. After processor services the signal, the interrupt is not detected again as no edge has occurred.

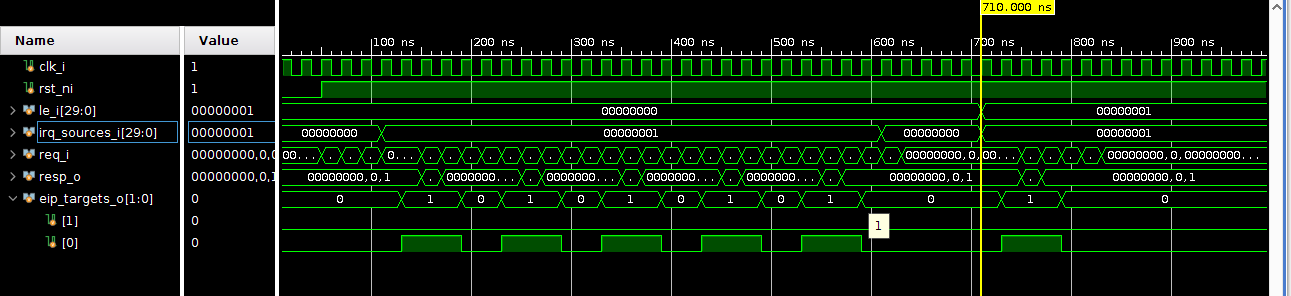


Figure : Simulation - Level / Edge trigger interrupt detection check

* Steps for level / edge trigger interrupt detection check
  + Enable interrupt source 1 and set priority. Clear the level / edge interrupt trigger signal for source 1.
  + Generate an extended interrupt signal. It is observed that the interrupt is detected again after the interrupt service is complete due to its detection type being level triggered.
  + When same check is performed for edge trigger, the interrupt is detected only once, as positive edge does not occur again.