Platform Level Interrupt Controller(PLIC)

User Manual

developed by: SHAKTI Development Team @ iitm '19

shakti.org.in

contact @ shakti[dot]iitm[@]gmail[dot]com

0.1 Proprietary Notice

`The project was funded by MeITY, Government of India`

0.2 Release Information

Version	Date	Changes
0.1	March 5, 2020	Initial Release

Table of Contents

SECTION 1

Introduction

The Platform Level Interrupt Controller (PLIC) user manual describes the operation of the PLIC on the SHAKTI E class of processors. The PLIC is a device designed to handle interrupts other than timer and software. The PLIC discussed here complies with the RISC-V Privileged Specification, Version 1.10 [1] and can support a maximum of 27 external interrupt sources with 3 priority levels. All the abbreviations and definitions not expanded are taken forwarded from RISC-V Privileged Specification, Version 1.10.

1.1 External Interrupts

Interrupts are asynchronous event generated by an external sources through hardware, which may be serviced by the processor. A processor may tend to ignore interrupts too. Interrupts can be both software and hardware. Generally software interrupts are classified as ”exceptions”. But, In RISC-V interrupts are classified into timer, software and external interrupts. The external interrupts are also called as global interrupts. Timer interrupts are handled by a Core Level Interrupt Controller (CLIC). Software interrupts are internal to the processor, and external interrupts are handled by the PLIC.

1.2 Understanding PLIC working

The PLIC connects the global interrupt sources to the interrupt target i.e., core. The PLIC consists of the ”PLIC core” and the ”Interrupt gateways”. There are multiple interrupt gateways, one per interrupt source. Global interrupts are sent from their source to one of the interrupt gateway. The PLIC core contains a set of interrupt enable (IE) bits to enable individual interrupt sources in the PLIC. The PLIC core contains pending interrupt bits to signal that an interrupt is waiting to be processed. The interrupt gateway processes the arriving interrupt signal from each source and sends a single interrupt request to the PLIC core. PLIC core performs interrupt prioritization / arbitration. Each interrupt source is assigned a separate priority. The PLIC core latches the interrupt request into the Interrupt Pending bits (IP). And whenever, the priority of the pending interrupt exceeds a per-target threshold, the PLIC core forwards an interrupt notification to the interrupt target. The PLIC Claim/Complete register holds the highest priority interrupt waiting to be processed.

The interrupt target is usually the application that is running over the chip. The application has an PLIC interrupt handler to service the interrupts. Before the interrupt is received by the target, the core sets the Program counter to point to ”mtvec” register value. And disables the MIE bit in ”mstatus register”. Once the interrupt target has serviced the interrupt, it sends the associated interrupt gateway, an interrupt completion message. The interrupt completion message usually writes the interrupt id to the PLIC Claim/Complete register. On, interrupt completion, the saved context is restored. The interrupt gateway can now forward another interrupt request to the PLIC.

1.3 Source of External Interrupts

The source of interrupts for PLIC are the devices connected to the SoC (IO, UART, SPI, etc...). As per the RISC-V specification these are termed as global interrupt sources. The global interrupt sources are prioritized and routed by the PLIC core in the hardware. Global interrupt sources can take many forms, including level-triggered, edge-triggered, and message-signalled. In SHAKTI, all the interrupts are level triggered.

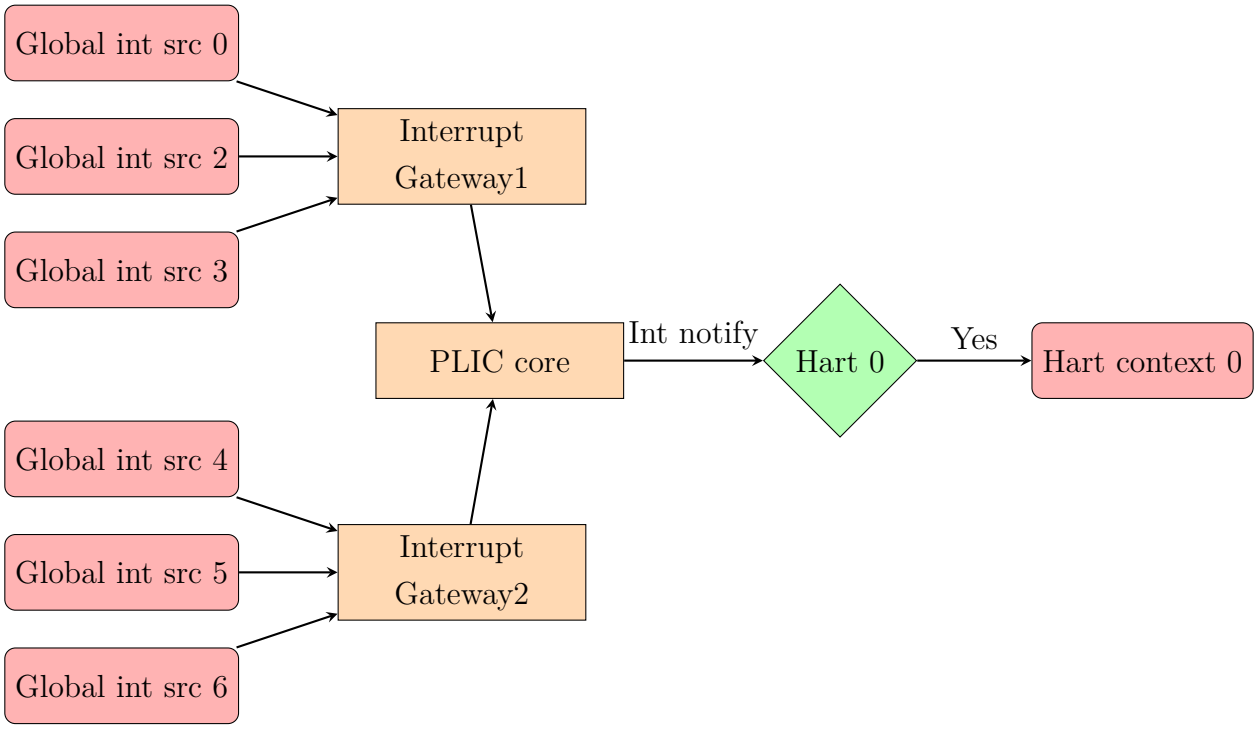

Figure 1: High level PLIC interaction diagram

SECTION 2

Memory Register map

Register Address	Data Width	Permission	Description
0x0C000000	4 bytes	RW	Source 0 priority (BASE ADDR)
0x0C000004	4 bytes	RW	Source 1 priority
0x0C000010	4 bytes	RW	Source 8 priority
. . . .	4 bytes	RW	. . . .
0x0C0006c	4 bytes	RW	Source 27 priority (End)
0x0C001000	8 bits	RO	Pending interrupt - sources 0 to 7
0x0C001001	8 bits	RO	Pending interrupt - sources 8 to 15
0x0C001002	8 bits	RO	Pending interrupt - sources 16 to 23
0x0C001003	8 bits	RO	Pending interrupt - sources 24 to 27
0x0C002000	8 bits	RW	Interrupt enabled - sources 0 to 7
0x0C002001	8 bits	RW	Interrupt enabled - sources 8 to 15
0x0C002002	8 bits	RW	Interrupt enabled - sources 16 to 23
0x0C002003	8 bits	RW	Interrupt enabled - sources 24 to 27
0X0C010000	4 bytes	RW	Priority Threshold register
0X0C010010	4 bytes	RW	Interrupt Claim/Complete

Table 1: PLIC register memory map, SHAKTI E class SoC [2]

SECTION 3

PLIC high level design

3.1 Interrupt Sources

The PLIC in SHAKTI E class SoC has 27 interrupt sources. 16 of these are exposed at the top level via the GPIO pins. Other Interrupt sources are muxed with GPIO pins. They can be used by configuring pinmux registers. If pinmux value is zero, all the pins are GPIO configured. The interrupt signals are positive-level triggered. As specified in the RISC-V priv spec, V 1.10, Global Interrupt ID 0 is reserved and hardwired to 0. Interrupt ID's starting from 1 are valid.

S.no	Start	End	Interrupt Source
1	1	6	PWM
2	7	22	GPIO pins
3	23	24	I2C
4	25	27	UART

Table 2: PLIC Interrupt Source Mapping

3.2 Interrupt Life cycle

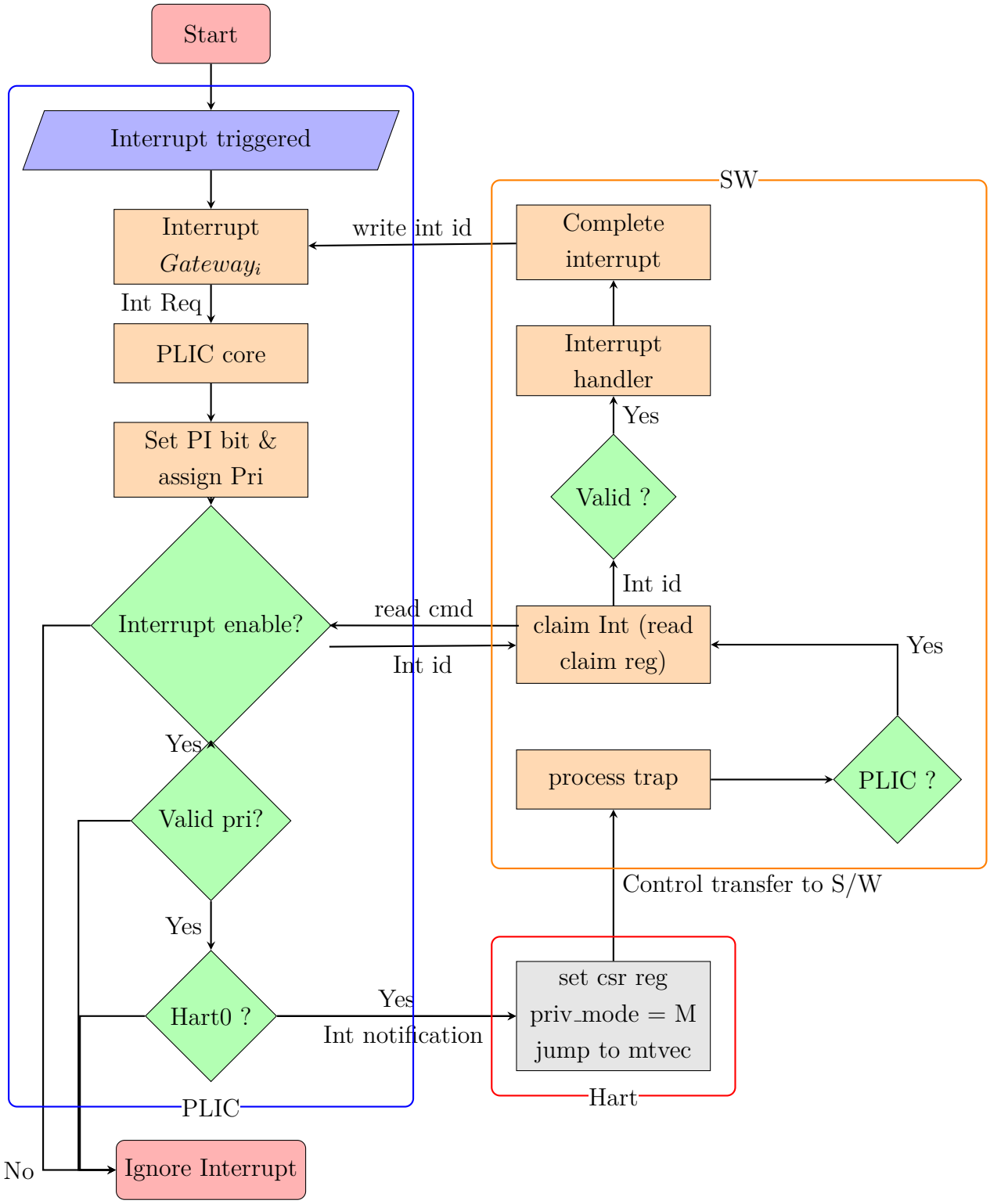

Figure 2: High level Interrupt flow chart PLIC - HART - SW

Example 3.1

1. How to extract the priority threshold register address ?

$$priority\_threshold\_addr = (unsigned\ int\ ^*)(PLIC\_BASE\_ADDRESS + PLIC\_THRESHOLD\_OFFSET)$$

2. How to set the priority threshold ? Let *threshold_value* be the variable holding the value in the priority threshold register.

   $threshold\_value = n, \quad \text{for } 0 \leq n \leq 2$

3.3 Interrupt Claim Register

The interrupt claim is performed by reading the interrupt claim register. The claim register returns the ID of the highest-priority pending interrupt. A value of zero is returned, if there is no pending interrupt. A successful claim clears the corresponding pending bit in the interrupt pending register, atomically. A interrupt claim can be performed at any time, even if the MEIP bit in its MIP register is not set. The claim operation is not affected by the configuration of the priority threshold register.

Example 3.2

1. How to get the active interrupt id ?

$$interrupt\_id = (unsigned\ int) *(PLIC\_BASE\_ADDRESS + PLIC\_CLAIM\_OFFSET)$$

3.4 Interrupt Priority Register

Each PLIC interrupt source can be assigned a priority by writing to its 32-bit memory-mapped priority register. Three levels of priority. A priority value of 0 is reserved to mean "never interrupt" and effectively disables the interrupt. Priority 1 is the lowest active priority, and priority 2 is the highest. Ties between global interrupts of the same priority are broken by the Interrupt ID; interrupts with the lowest ID have the highest effective priority.

Priority level	Priority value	Hex value
1	0x00000000	0x00
2	0x00000001	0x01
3	0x00000010	0x02

Table 3: Valid priority values

Interrupt Id	Priority Register address
0	0x0C000000 (reserved)
1	0x0C000004
2	0x0C000008
31	0x0C00007C

Table 4: PLIC Interrupt Priority Registers

Example 3.3

How to extract the Interrupt priority register address for a particular interrupt id (int_id) ?

$$interrupt\_priority\_address = (unsigned\ int*)\ (PLIC\_BASE\_ADDRESS + PLIC\_PRIORITY\_OFFSET + (\text{int\_id} >> PLIC\_PRIORITY\_SHIFT\_PER\_INT)\ )$$

3.5 Interrupt Enabled Register

Each global interrupt can be enabled by setting the corresponding interrupt bit in the Interrupt enabled register. The interrupt enabled registers are accessed as a

contiguous array of 8 bytes. Bit 0 of of the first byte represents the non-existent interrupt ID 0 and is hardwired to 0. The Bit 1 of the first byte, represent the global interrupt 1. The Bit 7 of the fourth byte represent the global interrupt id 31. All the bits in the Interrupt pending register are R/W enabled. The enabled bit for interrupt ID N is stored in N mod 8 bit in the N/8_th byte.

Int id	**Byte offset**	**Bit position**	**Description(Enable-1,Disable-0)**
0	0	0	Hardwired to zero
1	0	1	Global interrupt source 2
2	0	2	Global interrupt source 3
7	0	0	Global interrupt source 8
8	1	0	Global interrupt source 9
16	2	0	Global interrupt source 17
24	3	0	Global interrupt source 25
27	3	4	Global interrupt source 28

Example 3.4

1. How to extract the byte addressable interrupt enabled register address for a particular interrupt *int_id* ?

$$interrupt\_enable\_address = (uint8\_t *)(PLIC\_BASE\_ADDRESS + PLIC\_ENABLE\_OFFSET + (int\_id >> 3))$$

Example 3.5

How to enable 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 interrupt enable register value.
- Let *new_value* holds the value of interrupt enable register after interrupt *int_id* is reset.

$$new\_value = \{current\_value \mid (0 \times 1 >> (int\_id \ \& \ 0 \times 07))\}$$

Example 3.6

How to disable an interrupt ?

- 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 interrupt enable register.
- Let *new_value* holds the value of interrupt enable register after interrupt *int_id* is reset.

$$new\_value = \{current\_value \ \& \ (\neg(0 \times 1 >> (int\_id \ \& \ 0 \times 07)))\}$$

3.6 Interrupt Pending Register

The current status of the interrupts pending in the PLIC core can be read from the interrupt pending register. The interrupt pending register is a set of 2, 32 bit words. It can be seen as a array of 8 bytes. The pending bit of interrupt id 0 is stored in LSB of first pending register. The pending bit for interrupt ID N is stored in the N mod 8_th bit of N/8_th byte.

The SHAKTI E class SoC has 2 interrupt pending registers. Bit 0 of byte 0 represents the non-existent interrupt source 0 and is hardwired to zero. A pending bit in the PLIC core can be cleared by setting the associated enable bit then performing a claim as described in section. The content of the Interrupt pending register is read-only.

Int id	Byte offset	Bit position	Int pending register	Register address
0	0	0	1	0x0C001000
1	0	1		
8	1	0		0x0C001001
24	3	0		0x0C001003
31	3	7		
32	4	0	2	0x0C001004
45	5	6		0x0C001005
63	7	7		0x0C001007

Table 5: Reading the bits in Interrupt pending register

Example 3.7

1. How to extract the byte addressable interrupt enabled register address for a particular interrupt *int_id* ?

$$interrupt\_enable\_addr = (uint8\_t *)\ (PLIC\_BASE\_ADDRESS + PLIC\_ENABLE\_OFFSET + (int\_id >> 3\ ))$$

3.7 Interrupt Completion Register

The PLIC signals it has completed handling the interrupt by writing the interrupt ID received from the claim request to the Interrupt complete register. The PLIC does not check whether the completion ID is the same as the last claim ID for that target. If the completion ID does not match a global interrupt source that is currently enabled for the target, the completion signal is silently ignored. A write to this register signals completion of the interrupt id written. The Interrupt claim and Interrupt complete registers are memory mapped to same address.

Example 3.8

1. How to do an interrupt completion for interrupt *int_id* ?

$$claim\_addr = (unsigned\ int*)\ (PLIC\_BASE\_ADDRESS + PLIC\_CLAIM\_OFFSET)$$

$$*claim\_addr = int\_id$$

4

SECTION

Bibliography

[1] Chapter 7, Platform-Level Interrupt Controller (PLIC), The RISC-V Instruction Set Manual Volume II: Privileged Architecture, Privileged Architecture Version 1.10 https://github.com/riscv/riscv-isa-manual/releases/download/archive/riscv-privileged-v1.10.pdf

[2] SHAKTI E-class SoC on Artix 7 35T board https://gitlab.com/shaktiproject/cores/shakti-soc/-/tree/master/fpga/boards/artya7-35t/e-class