Design and Verification of Platform Level Interrupt Controller

Submitted by:

2019-FYP-33

Ahsan Ali	2019-EE-115
Ateeqa Afzal	2019-EE-101
Fatima Noor	2019-EE-104
Muhammad Shaheer	2019-EE-137

Supervised by: Dr. Ubaid Abdullah Fayyaz

Mr. Umer Shahid

Department of Electrical Engineering

University of Engineering and Technology Lahore

Contents

List of Figures

List of Tables

Abbreviations

PLIC	**P**latform **L**evel **I**nterrupt **C**ontroller
ie	**i**nterrupt **e**nable
Reg Map	**Reg**ister **Map**ping
FPGA	**F**ield **P**rogrammable **G**ate **A**rrays
CPU	**C**entral **P**rocessing **U**nit

Abstract

Whenever we think that someone has teased us while we are working, that teasing is an interrupt in our work for us. Exactly the same thing happens in the life of computer. While Processor is executing its routine instructions, we interrupt it and ask for the work that is urgent than its routine. Here Interrupt controller comes into being. There are two type of programs in the world of Embedded coding, polling based programs and interrupt based programs. Obviously interrupt based programs are faster than polling based programs because they have low latency and have some handlers to be handle by processor when it is called. We will see that how can we implement Interrupt controller for a large number of interrupts for RISC-V processor.

Chapter 1

Introduction

An Embedded System communicates with the outside world through its input/output devices. The computer can acquire information through input devices, and it can show information through output devices. Output devices also allow the computer to manipulate its environment. An embedded system differs from a standard computer system by having a close connection to the outside world. The difficulty is that, in the majority of cases, software runs considerably more quickly than hardware. For instance, it might take the software only 1 μs to ask the hardware to clear the LCD, but the hardware might take 1 ms to complete the command. During this time, software may carry out ten thousands of instructions. As a result, for an embedded system application to be successful, the synchronisation between the executing software and its external environment is necessary.

This synchronization is somehow managed by sending an interrupt. An interrupt is an unexpected event that occurs during normal working of a system. It is the automatic transfer of software execution in response to a hardware event that is asynchronous with the current software execution.

In computer architecture mostly external devices sends interrupt. External devices are comparatively slower than CPU. So if there is no interrupt CPU would waste a lot of time waiting for external devices to match its speed with that of CPU. This decreases the efficiency of CPU. Hence, interrupt is required to eliminate these limitations.

The processor services the interrupts. In RISC-V interrupts are classified into timer, software and external interrupts. The external interrupts are also called as global interrupts. Timer and software interrupts are handled by a Core Local Interrupt (CLINT). External interrupts are handled by the **PLIC**.

Platform level interrupt controller is shortly known as PLIC. The PLIC connects global interrupt sources, which are usually I/O devices, to interrupt targets, which are usually hart contexts. The I/O devices need attention so it will interrupt one of the core. For example human types character keyboard will send interrupt to any of the core through PLIC and as a result that core will talk directly to the keyboard. It will run an interrupt handler. A trap will occur and then core will make interrupt handler

to run and ask keyboard what character was typed. When the operation is finished the core will return to its original operation and resume its work.
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 these in the core interrupt pending bits (IP). Each interrupt source is assigned a 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.

Chapter 2

Problem Statement

Interrupt is referred as an input signal that has the highest priority for hardware or software events that requires immediate processing of an event. During the early days of computing, the processor had to wait for the signal to process any events. The processor should check every hardware and software program to understand if there is any signal to be processed. The two methods used at early time were Blind Cycle Method and Polling Base Method.

When an I/O operation is performed, the device hardware incurs some delay in responding to the software request. This delay corresponds to the time interval starting from the time instance when software makes the I/O operation request, till the device hardware has finished the operation. If this delay is highly predictable due to the fact that any variations in this delay are relatively small then it is possible that the software can initiate a new I/O request after fixed delay. This is precisely what is done in **Blind Cycle Synchronization**, where the software waits for a fixed time interval assuming that the I/O device hardware has completed the operation within this time interval.

The unnecessarily large delays incurred by blind cycle method can be reduced by checking the status of the device hardware either continuously or periodically. Using this status information the software can eliminate any excessive delays and waits only for the required duration. Status checking of the device hardware, by the software, is also termed as **Polling Base Method**. This method would consume a number of clock cycles and makes the processor busy. Just in case, if any signal was generated, the processor would again take some time to process the event, leading to poor system performance.

A new mechanism was introduced to overcome this complicated process. In this mechanism, hardware or software will send the signal to a processor, rather than a processor checking for any signal from hardware or software. The signal alerts the processor with the highest priority and suspends the current activities by saving its present state and function, and processes the interrupt immediately, this is known as Interrupt Service Routine (ISR). As it doesn't last long, the processor restarts normal activities as

soon as it is processed.

Interrupt is sent to a processing unit to check proper working of a microprocessor. As mentioned earlier the physical working of a keyboard, mouse or any disk is interpreted by processor. The core will stop its working and allow the interrupt to be processed. When gateway is notified about the completion of interrupt, next interrupt is processed if any. Else it core will resume its work.
This platform level interrupt controller is widely used in embedded system. The designers uses interrupt service to test the appropriate working of their processors.

Chapter 3

Literature Review

Interrupts are Asynchronous events generated by a external source via hardware. In RISC-V interrupts are classified into timer, software and external interrupts. The external interrupts are also called as global interrupts. Timer and software interrupts are handled by a Core Local Interrupt. External interrupts are handled by the PLIC.

3.1 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 interrupt gateway processes the arriving interrupt signal from each source and sends a single interrupt request to the PLIC core. The PLIC core contains a set of interrupt enable 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. Also, PLIC core performs interrupt prioritization. Each interrupt source is assigned a separate priority. The PLIC core latches the interrupt request into the Interrupt Pending bits. 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 register holds the highest priority interrupt waiting to be processed. On interrupt completion the interrupt Gateways can now send another interrupt request to the PLIC.

3.1.1 Operation Parameters

Register blocks that perform PLIC operation parameters are Interrupt Priorities registers for the selection of interrupt priority for each interrupt source. The second one is Interrupt Pending Bits register for the interrupt pending status of each interrupt source. The third one is Interrupt Enables register to perform the enablement of interrupt source of each context. The fourth one is Priority Thresholds register to select

the interrupt priority threshold of each context. The fifth one is Interrupt Claim register to acquire interrupt source ID of each context. And finally the Interrupt Completion register to send interrupt completion message to the associated gateway.

3.1.2 Memory Register Map

The base address of PLIC Memory Map is platform implemented and its width is 32-bit.

Register Adress.	Data Width	Description
0x0C000000	4 bytes	Source zero priority(base Addr)
0x0C000004	4 bytes	Source 1 priority
. .		
0x0C002003	8 bytes	Interrupt Enabled Source 24 to 27
0x0C000000	4 bytes	Priority Threshold register
0x0C010010	4 bytes	Interrupt claim

3.2 Interrupt Prioritues

Interrupt priorities are small unsigned integers, with a platform-specific maximum number of supported levels. The priority value 0 is reserved to mean "never interrupt", and interrupt priority increases with increasing integer values. Each global interrupt source has an associated interrupt priority held in a memory-mapped register. Different interrupt sources need not support the same set of priority values. A valid implementation can hardwire all input priority levels. 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. 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. The base address of Interrupt Source Priority block within PLIC Memory Map region is fixed at 0x000000.

PLIC register block name	Register Block size in byte	Function
Interrupt Source Priority	1024*4 = 4096(0x1000) bytes	Interrupt Source Priority 0 to 1023

3.2.1 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 8th bit of N/8th byte. The PLIC 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.

3.3 Interrupt Enablers and Priority Thresholds

Each global interrupt can be enabled by setting the corresponding bit in the enables register. The enables registers are accessed as a contiguous array of 32-bit registers, packed the same way as the pending bits. Bit 0 of enable register 0 represents the non-existent interrupt ID 0 and is hardwired to 0. PLIC has 15872 Interrupt Enable blocks for the contexts.In PLIC a large number of potential IE bits might be hard-wired 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.

PLIC provides context based threshold register for the settings of a interrupt priority threshold of each context. The threshold register is a WARL field. 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.

3.3.1 Interrupt Completion

When PLIC signals completes executing an interrupt handler by writing the interrupt ID it received from the claim to the claim 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 an interrupt source that is currently enabled for the target, the completion is silently ignored. 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.

Chapter 4

Project Overview and Objectives

The Program Level interrupt controller deals with interrupts generated by the peripherals and the processors, handles the interrupt priorities, and delegates the execution to a processor. An interrupt request is an asynchronous signal that is typically triggered by an I/O device which needs to be serviced.

The PLIC in our case has finite interrupt sources. Some of these will be 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. And if one pin goes to the I2C. The source of interrupts for PLIC are the devices connected to the SoC (GPIO, UART, I2C, etc...). As per the RISC-V specification these are termed as global interrupt sources. Global interrupt sources can take many forms, including level-triggered and edge-triggered. In our case, all the interrupts will be positive edge triggered. Our concept will be more clear by looking onto the above fig.

4.1 Design Management Challanges

4.1.1 Design Flexibility and Complexity

- Potentially hundreds and even thousands of registers in Memory Mapped Management Interface.
- Interface Complexity discourages design changes.
- Documentation is time-consuming and error-prone.

4.1.2 Interface Design Goals

- Keep memory map as small as possible.
- Maintain an intuitive, logical arrangement of registers.
- Avoid need for manual maintenance of interface code.

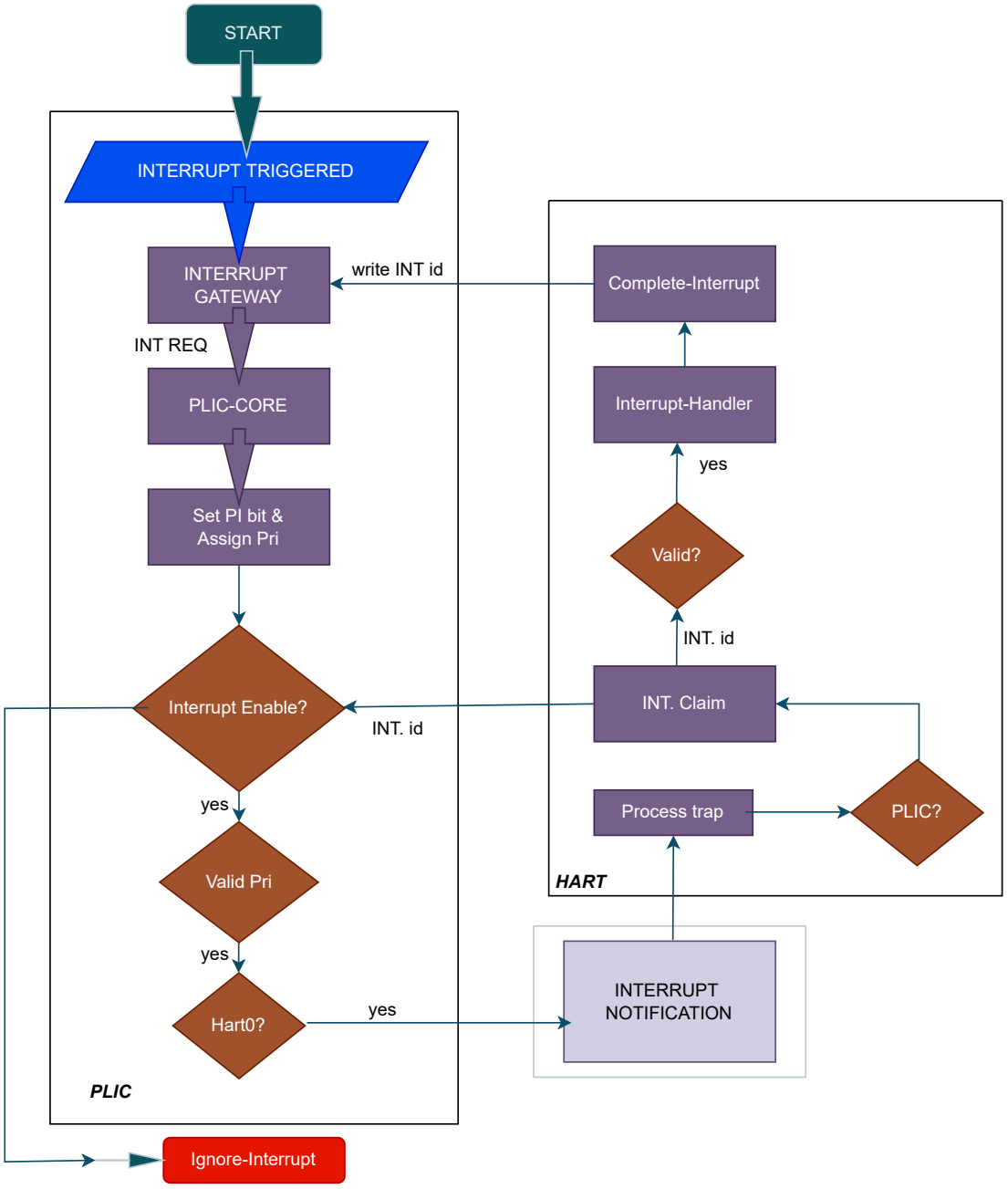

FIGURE 4.1: Interrupt Controller Overview

4.2 Design Management Solutions

4.2.1 Dynamic Memory Map

- Easily adapt to wide range of parameters.
- Automate practical register arrangement.
- Automate documentation of memory map.

4.2.2 No. of Registers

- Re-Use Read ID register as “Interrupt Claim”.
- Re-Use Write ID register as “Interrupt Complete”.

4.3 PLIC Design Goals

- Easy integration with external bus interfaces.
- Support user defined number of Interrupt Sources and Targets.
- Enabling and disabling of individual interrupt sources per target.
- Full Priority Level and Priority Threshold support.
- Low latency handling of queued interrupt requests.
- Programmable depth queue of pending interrupts.

Chapter 5

Project Development Methodology/Architecture

This chapter describes the project development methodology and architecture for the RISC-V platform-level interrupt controller (PLIC), which prioritizes and distributes global interrupts in a RISC-V system.
The PLIC connects global interrupt sources to interrupt targets. The PLIC can be distributed into three modules that are named as PLIC gateway, PLIC register map and PLIC target as explained in the block diagram in figure 7.1.

5.0.1 PLIC GATEWAY

The figure 5.1 shows the block diagram of module of the PLIC gateway.

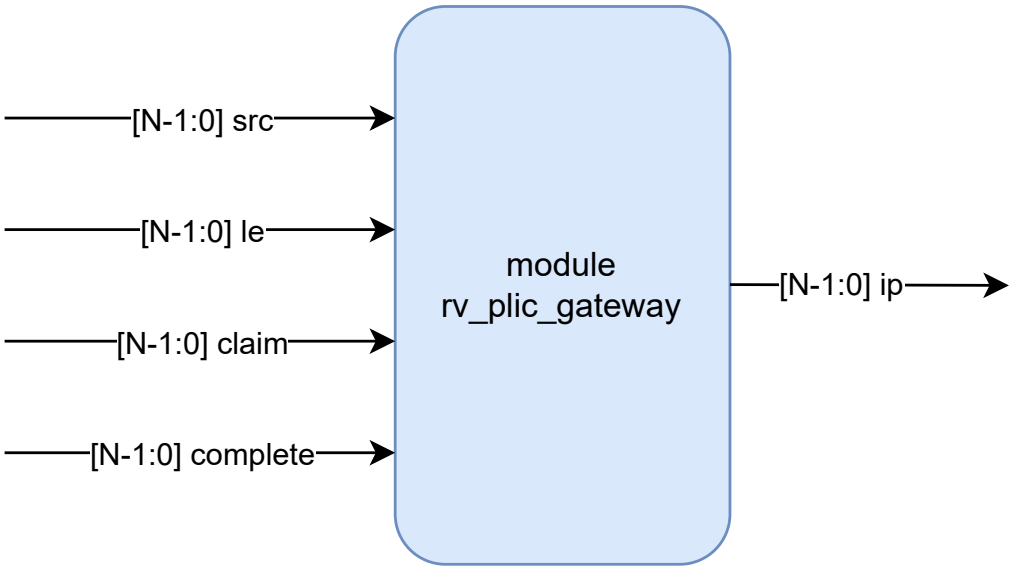

FIGURE 5.1: PLIC Gateway Module.

The interrupt gateways in the platform level interrupt controller 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. At most one interrupt request per interrupt source can be pending in the PLIC core at any time and it can be indicated by setting the source's interrupt pending bit. After receiving notification that the interrupt handler servicing the previous interrupt request from the same source has completed, the gateway forwards a new interrupt request to the PLIC core. 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. If the interrupt is level-triggered and the interrupt is still asserted, a new interrupt request will be forwarded to the PLIC core on receiving an interrupt completion message. The gateway does not have the facility to retract an interrupt request once forwarded to the PLIC core. If a level-sensitive interrupt source de-asserts 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. If the global interrupt source was 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 has been received. If the gateway has a pending interrupt counter, the counter will be decremented when the interrupt request is accepted by the PLIC.

5.0.2 PLIC TARGET

The figure 5.2 shows the block diagram of module of the PLIC Target.

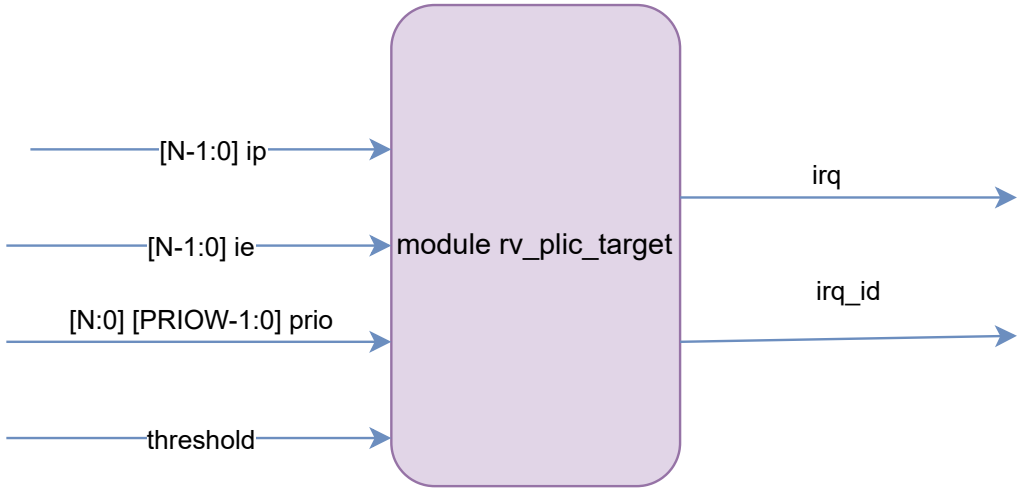

FIGURE 5.2: PLIC Target Module.

The PLIC target deal with the interrupt pending bit, interrupt enable bit, interrupt priority and threshold.
The current status of the interrupts pending in the PLIC core can be read from the interrupt pending register.
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. Each global interrupt can be enabled by setting the corresponding bit in the enables register. 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 a target. IE bits are WARL fields that can be hardwired to either 0 or 1. Bit 0 of enable register 0 represents the non-existent interrupt ID 0 and is hardwired to 0.
The PLIC supports interrupt priorities, that is each PLIC interrupt source can be assigned a priority by writing to its memory-mapped source priority register. A priority value of 0 is reserved to mean never interrupt and effectively disables the interrupt. Priority 1 is the lowest active priority while the maximum level of priority is defined by the max priority parameter. Ties between global interrupts of the same priority are broken by the Interrupt ID; interrupts with the lowest ID have the highest effective priority.
The threshold priority level is set via the Priority Threshold Register. An interrupt line with a priority less than the threshold, is masked. As an example, a threshold value of zero permits all interrupts with non-zero priority.

5.0.3 PLIC Register Map

The figure 5.3 shows the block diagram of module of the PLIC Target.
The register map for the PLIC control registers is shown in table 5.1

Distribute the project goals into smaller objectives/modules and outline deliverables for each objective. Explain the modules of the project through a system-level block diagram. Students may also mention tools, technologies, and suitability of the method(s) to be employed with justification. In case of a research problem, outline the approaches that will be investigated in the project. $(2-3$ pages)

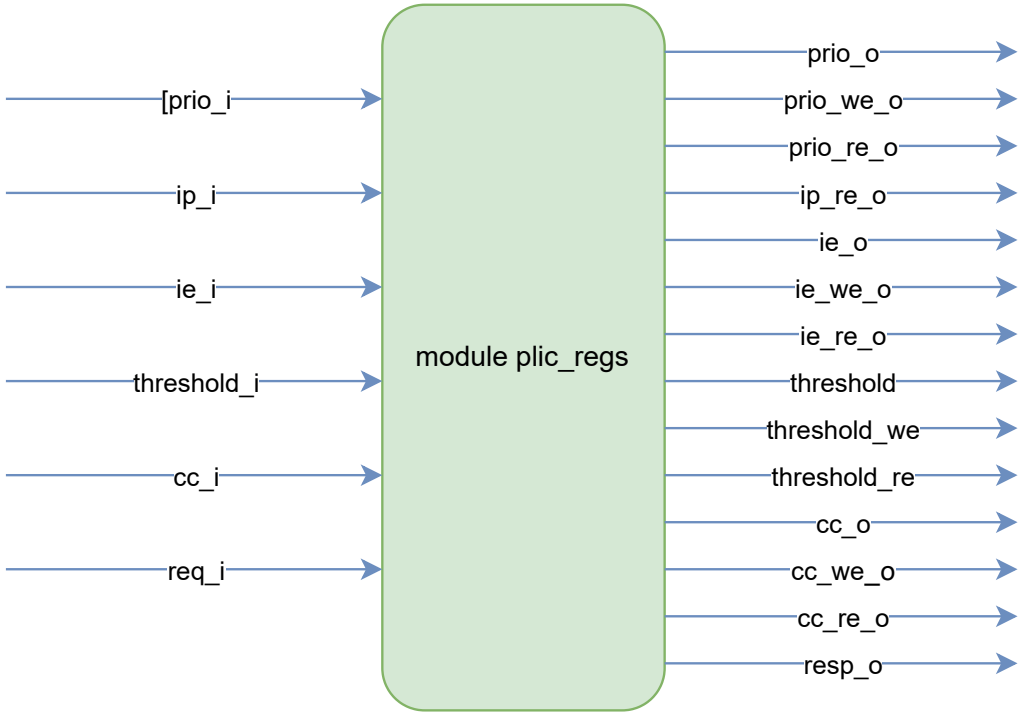

FIGURE 5.3: PLIC Target Module.

Register-Name	Offset(hex	Size(Bits)	Reset(hex	Description
source priority-0	0X0	24	0X0	Register holds the priority value of the respective interrupt source
source priority-1	0X4	24	0X0	
...	...	...	...	
source priority-31	0X7C	24	0X0	
source pending	0X1000	32	0X0	Register holds the pending interrupt bits for upto 32 sources in a single register
target enables	0X2000	32	0X0	Register holds the interrupt enable bits for upto 32 sources in a single register
target threshold	0X200000	32	0X0	Register holds the priority threshold of the respective target
target claim	0X200004	32	0X0	Register holds interrupt claim/completion information for the respective target

TABLE 5.1: PLIC Register Map for all configuration registers

Chapter 6

Project Milestones and Deliverables

The Gantt Chart shown in figure 6.1 explains the milestones of the project of Design and Verification of Platform Level Interrupt Controller.

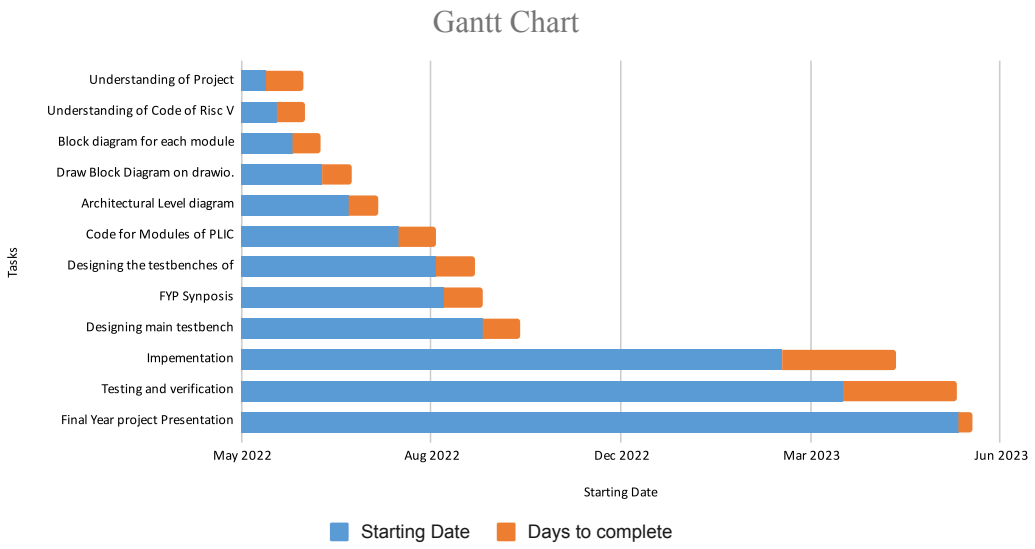

FIGURE 6.1: Gantt Chart for PLIC project

Chapter 7

Block Diagram

As interrupts come from sources (from the same or different sources), we also tell whether the interrupt is **Level or edge triggered**? Then gateway passes these interupts to target in the form of ip (interrupt pending).

Target then segregates the interrupts with respect to their priority. Then finally, using valid-ready interface interrupt is sent to RISC-V processor to service it. High priority interrupt is serviced first.

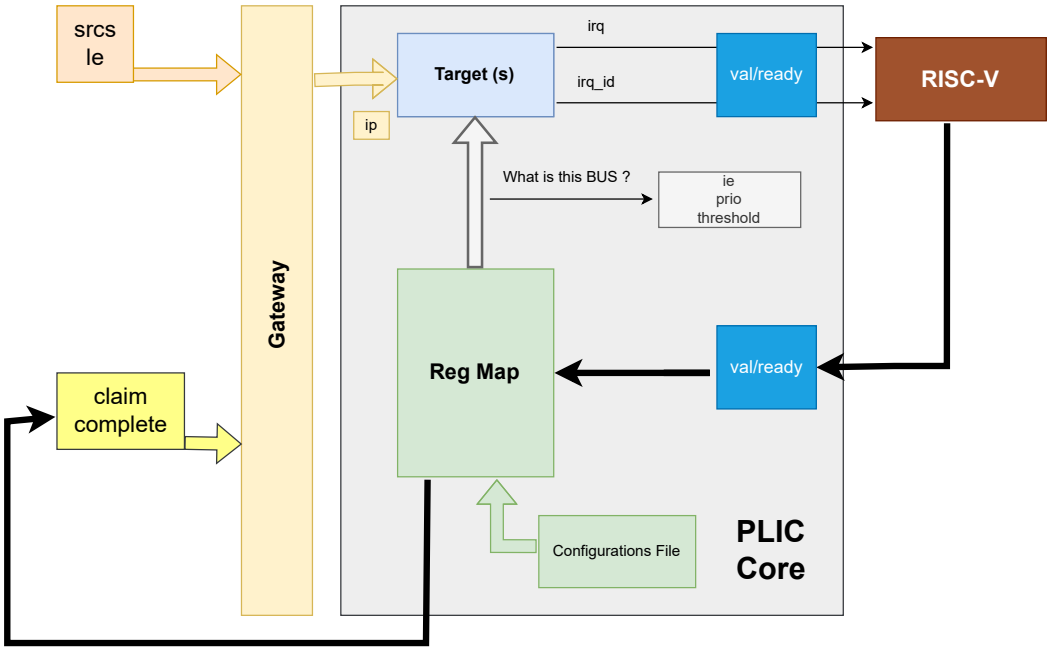

FIGURE 7.1: Block Diagram.

As we see in the figure 7.1, there is also a block Reg Map (Register Map). It is used to store a particular interrupt is enabled or not, what is its priority, what is the threshold of the target to take that interrupt? etc. We program all this stuff using configuration file. At last, Processor also tells PLIC core that particular interrupt is taken and PLIC takes action to mark it as complete.

Chapter 8

Flow Chart

Once interrupt source triggers the interrupt, it then goes to gateway and then gateway will request that interrupt to PLIC Core which is shown in the Figure 8.1.

After the interrupt is requested to PLIC core, it will then **alert/notify** target(s). Now the important step comes, after the target has received the interrupt, it will send a signal back to PLIC core that this interrupt is claimed (Also the claim ID). Receiving the claim from target, PLIC will send claim response that this interrupt should be serviced or not. Then the interrupt handler services the that particular interrupt.

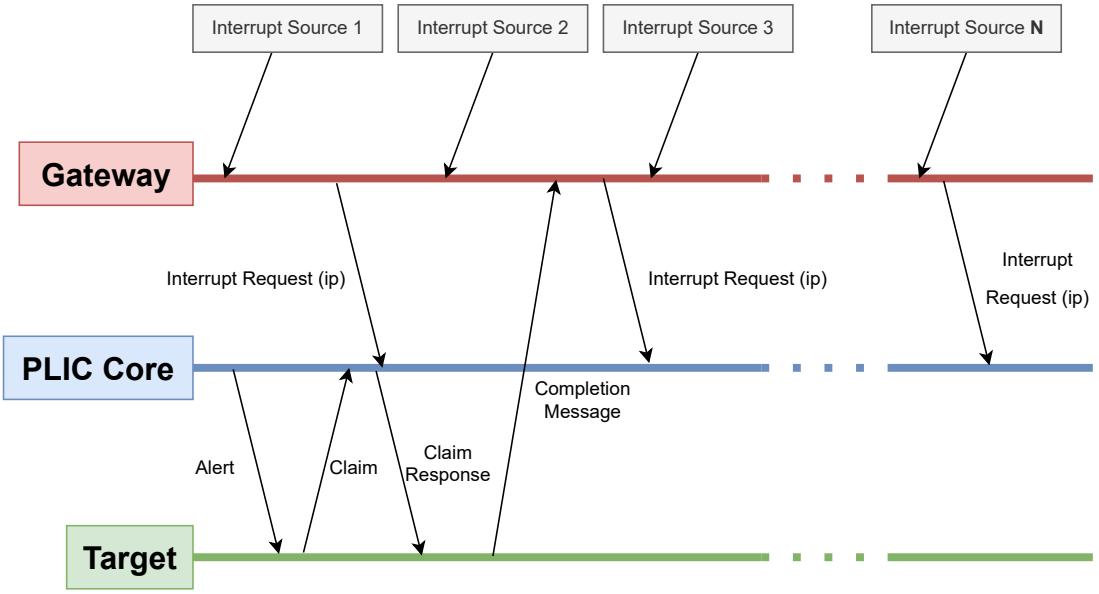

FIGURE 8.1: Flow Chart.

Finally, the interrupt completion message will be sent to gateway that this interrupt is completed and no need to send any other interrupt from that source. So is the case with all other pending interrupts from other sources.

Chapter 9

Work Division

The work division among the four members is given below:

9.0.1 Fatima's Work

She is responsible to Design and Verify the **Gateway(s)**. This is a block or a device which will convert the interrupts from different sources to interrupt requests based on their triggering type (Level/Edge). It will also take care of interrupt claim and complete message from RISC-V.

9.0.2 Shaheer's Work

He is responsible to Design and Verify the **Target**. This is a block or a device which will prioritize the interrupt requests to be sent to RISC-V. It will be based on mainly two algorithms. User will have to decide the algorithm for prioritization.

9.0.3 Ahsan's Work

He is responsible to Design and Verify the **Register Map (Reg Map)**. This module is a set of registers which will be programmed externally to give data to almost every device inside the PLIC. For example, If Gateway wants ie (interrupt enable) signal, then it will take from register map. If **Target** wants priorities of the enabled interrupts, then it will take from register map. And so on.

9.0.4 Ateeqa's Work

She is responsible to interface and merge all the modules to a new device which is eventually going to be the PLIC Core to communicate among all the devices like Gateway, Target, Register Map etc.

Chapter 10

Costing

This project requires quite low budget becaue it is 60 to 70 percent software based. The only hardware required is FPGA.

Sr No.	Device Name	Software Name	Cost
1	Xilinx's Nexys A7 FPGA	xc7a100tcsg324-1	64000 Rs.

Rest of the 30 to 40 percent project will be hardware based where Verification of the project will be done on test benches as well as on FPGA. The total cost for the project is rounded off (Due to some other microcontrollers) is coming out to be 70,000 PKR.

References

[1] Wei Chipin, Li Zhao lin, Zheng Qingwei, Ye Jianfei and Li Shenglong, "Design of a configurable multichannel interrupt controller," 2010 Second Pacific-Asia Conference on Circuits, Communications and System, 2010, pp. 327-330, doi: 10.1109/PACCS.2010.5626805.

[2] C. Y. Sia, B. A. Rosdi and M. C. Lee, "Synchronous design of 8259 Programmable Interrupt Controller," 2011 IEEE International Conference on Computer Applications and Industrial Electronics (ICCAIE), 2011, pp. 195-200, doi: 10.1109/ICCAIE.2011.6162130.

[3] Y. Liu, W. Zhang, J. Zhou, Y. Wang, S. Wang and L. Zhang, "A Design of Encoding Arbitration and Interrupt Request for Dynamic Reconfigurable High-Speed Serial Bus in Cyber Physical System," 2019 International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), 2019, pp. 627-634, doi: 10.1109/iThings/GreenCom/CPSCom/SmartData.2019.00122.

[4] N. Ito, Y. Oosako, N. Ishiura, H. Kanbara and H. Tomiyama, "Binary Synthesis Implementing External Interrupt Handler as Independent Module," 2017 International Symposium on Rapid System Prototyping (RSP), 2017, pp. 92-98.

[5] A. Tumeo et al., "An Interrupt Controller for FPGA-based Multiprocessors," 2007 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation, 2007, pp. 82-87, doi: 10.1109/ICSAMOS.2007.4285737.

[6] K. F. Baker and P. B. Crilly, "A priority event monitor for an interrupt-driven microprocessor," IEEE Proceedings of the SOUTHEASTCON '91, 1991, pp. 905-909 vol.2, doi: 10.1109/SECON.1991.147891.

[7] K. F. Baker and P. B. Crilly, "Priority event monitoring for interrupt-driven computer," Conference Record of the 1992 IEEE Industry Applications Society Annual Meeting, 1992, pp. 1751-1755 vol.2, doi: 10.1109/IAS.1992.244521.

[8] Zijia Guo, Teng Wang, Xin-An Wang and Ziyi Hu, "Design of an optimized low-latency interrupt controller for IMS-DPU," 2013 IEEE 10th International Conference on ASIC, 2013, pp. 1-4, doi: 10.1109/ASICON.2013.6811840.

[9] A. de Gloria, P. Faraboschi and M. Olivieri, "A self timed interrupt controller: a case study in asynchronous micro-architecture design," Proceedings Seventh Annual IEEE International ASIC Conference and Exhibit, 1994, pp. 296-299, doi: 10.1109/ASIC.1994.404555.

[10] De Alba, M., Andrade, A., González, J., Gómez-Tagle, J., & García, A. D. (2007). FPGA design of an efficient and low-cost smart phone interrupt controller. Latin American applied research, 37(1), 59-63.

[11] Harris, S., 2022. Digital design and computer architecture. Amsterdam: Morgan Kaufmann.

[12] TAHIR, M., 2020. ARM MICROPROCESSOR SYSTEMS. 1st ed. [S.l.]: CRC PRESS.

[13] Hennessy, J., Patterson, D. and Arpaci-Dusseau, A., n.d. Computer architecture.

[14] Bing.com. 2022. Platform Level Interrupt Controller. [online] Available at: ¡https://www.bing.com/ck/a?!&&p=8315da1b9e1496e5JmltdHM9MTY2MzgwND gwMCZpZ3VpZD0yZTgyNmJiNS04Njk4LTZlNjktM2M4Zi03YWRiODc5OTZmMGYmaW5za WQ9NTI5MA&ptn=3&hsh=3&fclid=2e826bb5-8698-6e69-3c8f-7adb87996f0f&u=a1aHR0cHM6Ly9 yb2Fsb2dpYy5jb20vcG9ydGZvbGlvL3BsYXRmb3JtLWxldmVsLWludGVycnVwdC1jb250cm9sbC [Accessed 3 July 2022].

[15] Chromitem-soc.readthedocs.io. 2022. 18. Platform Level Interrupt Controller (PLIC) — Chromite M SoC Manual 0.11.0 documentation. [online] Available at: ¡https://chromitem-soc.readthedocs.io/en/latest/plic.html#¿ [Accessed 3 July 2022].