

Computer Architecture

8259 PIC Project

The project aims to design and implement a Programmable Interrupt Controller (PIC) based on the 8259-architecture using Verilog hardware description language. The 8259 PIC is a crucial component in computer systems responsible for managing and prioritizing interrupt requests, facilitating efficient communication between peripherals and the CPU.

# Team Members:

|  |  |
| --- | --- |
| Name | ID |
| Ahmed Khaled Abdelmaksod Ebrahim | 2000218 |
| Adham Khaled Abd El Maqsoud | 2000066 |
| Karim Ibrahim Saad Abd-Elrazek | 2001118 |
| Eslam Mohamed Marzouk Abdelaziz | 2000252 |
| Mahmoud Abdelraouf Mahmoud | 2001436 |
| Ahmed Mohammed Bakr Ahmed | 2000037 |

# Table Of Contents:

Table of Contents

[1. Team Members: 2](#_Toc154377353)

[2. Table Of Contents: 2](#_Toc154377354)

[3. Key Features: 3](#_Toc154377355)

[4. Block Diagram: 4](#_Toc154377356)

[5. Signals Description: 5](#_Toc154377357)

[a) Data Buffer: 5](#_Toc154377358)

[b) R\W Logic: 5](#_Toc154377359)

[c) Cascade Controller: 6](#_Toc154377360)

[d) IMR: 7](#_Toc154377361)

[e) In Service Register: 7](#_Toc154377362)

[f) Priority Resolver: 8](#_Toc154377363)

[g) Interrupt Request Register: 9](#_Toc154377364)

[h) Control Logic: 10](#_Toc154377365)

[6. Testing Strategy: 12](#_Toc154377367)

[7. Test Benches Snapshots: 12](#_Toc154377368)

[8. Team Tasks: 17](#_Toc154377369)

[9. References: 17](#_Toc154377370)

[10. GitHub Repo: 17](#_Toc154377371)

# Key Features:

1. **8259 Compatibility:** The Verilog implementation will closely emulate the behavior and features of the classic 8259 PIC, ensuring compatibility with existing systems and software.
2. **Programmability:** The project will include support for programming interrupt priorities and modes, allowing users to configure, using Command Words (ICWs) and Operation Command Words (OCWs), the PIC according to their specific requirements.
3. **Cascade Mode:** Implementing the cascade mode, where multiple PICs can be interconnected to expand the number of available interrupt lines, enhancing the scalability of the design.
4. **Interrupt Handling:** Efficient handling of interrupt requests, including prioritization and acknowledgment mechanisms, to ensure a timely and accurate response to various events.
5. **Interrupt Masking:** Implement the ability to mask/unmask individual interrupt lines to control which interrupts are currently enabled.
6. **Edge/Level Triggering:** Support both edge-triggered and level-triggered interrupt modes to accommodate different types of peripherals.
7. **Fully Nested Mode:** Implement Fully Nested Mode, allowing the PIC to automatically set the priority of the CPU to the highest priority interrupt level among the currently serviced interrupts.
8. **Automatic Rotation:** Extend the priority handling mechanism to support automatic rotation even in scenarios where lower-priority interrupts are being serviced.
9. **EOI:** Implement the EOI functionality, allowing the PIC to signal the end of interrupt processing to the CPU.
10. **AEOI:** Implement the EOI functionality, allowing the PIC automatically to signal the end of interrupt processing to the CPU.
11. **Reading the 8259A Status:** Implement the capability to read the status of the 8259A PIC.
12. **Simulation and Testing:** Comprehensive testbench development for simulating and validating the functionality of the Verilog-based 8259 PIC. This includes testing various interrupt scenarios and ensuring proper interaction with other system components.
13. **Documentation:** Thorough documentation detailing the design specifications, module functionalities, and usage guidelines to facilitate understanding and future development.

# Block Diagram:

This is our neat sketch for the block diagram we worked on.

フ ~ = フ ィ ) つ 
73rd 
フ ~ ノ 
叫 / 
「 す っ - 0 ー っ 
い 一 四 
ー フ 0 
を み ツ メ 
つ 一 0 
っ 9 ア 0 
/ 0 

Figure 1. neat sketch for block diagram.

# Signals Description:

In this section you will find each block as a header encapsulating the signals with description.

## Data Buffer:

A graph paper with a graph and a graph

Description automatically generated

Figure 2. Data Buffer

|  |  |
| --- | --- |
| Signal | Description |
| data\_inside | **inout wire [7:0]: register represents input data** |
| data\_outside | **inout wire [7:0]: register represents output data** |
| rd | **input wire: write operation active low** |
| wr | **input wire: read operation active low** |

## R\W Logic:

A diagram of a diagram

Description automatically generated

Figure 3. R\W Logic

|  |  |
| --- | --- |
| Signal | Description |
| Read | **Read control signal.** |
| write | **Write control signal** |
| A0 | **Address bit 0.** |
| CS | **Chip Select control signal.** |
| dataBuffer | **8-bit data input.** |
| write\_flag\_ACK | **Acknowledgment signal for a write operation.** |
| OCW3\_change\_ACK | **Acknowledgment signal for a change in OCW3.** |
| write\_flag | **Output signal indicating a write operation has been acknowledged.** |
| ICW1 | **Initialization Command Word 1 (8 bits).** |
| ICW2 | **Initialization Command Word 2 (8 bits).** |
| ICW3 | **Initialization Command Word 3 (8 bits).** |
| ICW4 | **Initialization Command Word 4 (8 bits).** |
| OCW1 | **Operation Command Word 1 (8 bits).** |
| OCW2 | **Operation Command Word 2 (8 bits).** |
| OCW3 | **Operation Command Word 3 (8 bits).** |
| read\_cmd\_to\_ctrl\_logic | **command signal for control logic to tell it to active read status from IRR OR ISR registers** |
| read\_cmd\_imr\_to\_ctrl\_logic | **Command signal for control logic to activate read status from IMR registers.** |
| OCW3\_change | **Output signal indicating a change in OCW3.** |

## Cascade Controller:

A diagram of a company

Description automatically generated

Figure 4: Cascade Controller

|  |  |
| --- | --- |
| Signal | Description |
| CAS | **Input/Output: Cascade control lines** |
| SP | **Input: Selects between MASTER and SLAVE modes.** |
| ICW3 | **Input: ICW3 signal, used for configuration (In case of Slave).** |
| control\_signal | **Input: Selects between MASTER and SLAVE modes.** |
| desired\_slave | **Input: Desired slave ID in case of MASTER mode.** |
| flag\_ACK | **Input: Acknowledge flag indicating a successful flag update.** |
| EOI | **Input: End of interrupt flag from control logic to put 3'bzzz on cascade lines (In case of Master)** |
| control\_signal\_ack | **Output: Acknowledge to control signal flag.** |
| flag | **Output: Flag indicating if it's the desired slave.** |
| SP\_to\_control | **Output: send SP signal to control logic to know the state of pic (master or slave)** |

## IMR:

A close-up of a graph

Description automatically generated

Figure 5: IMR

|  |  |
| --- | --- |
| Signal | Description |
| OCW1 | **OCW1 commands to know which bits are masked. conected to the R/D logic.** |
| readIMR | **To put the IMR\_reg into the internal data lines, connected to the control logic.** |
| IMR\_reg | **IMR (status) register, connected to IRR.** |

## In Service Register:



Figure 6: ISR

|  |  |
| --- | --- |
| Signal | Description |
| toSet | **Input: Signals indicating which interrupts to service (IR0-IR7)** |
| readPriority | **Input: Signal to read values from toSet** |
| readIsr | **Input: Signal to output value of isrReg to IsrRegValue** |
| sendVector | **Input: Signal to output vectorTable value to dataBuffer** |
| zeroLevelIndex | **Input: Signals indicating IRx with highest priority** |
| ICW2 | **Input: Initialization Command Word 2** |
| ICW4 | **Input: Initialization Command Word 4** |
| secondACK | **Input: Second acknowledge signal** |
| changeInOCW2 | **Input: Signal indicating change in OCW2** |
| OCW2 | **Input: Operation Command Word 2** |
| INTIndex | **Output: Signals indicating which interrupts to service (IR0-IR7)** |
| dataBuffer | **Output: Value of isrReg to the dataBuffer** |
| isrRegValue | **Output: Value of isrReg to the PriorityResolver** |
| resetedIndex | **Output: Signal indicating end of interrupt mode** |
| sendVectorAck | **Output: Signal to acknowledge sendVector** |

## Priority Resolver:

A close-up of a grid with text

Description automatically generated

Figure 7: Priority Resolver

|  |  |
| --- | --- |
| Signal | Description |
| freezing |  |
| IRR\_reg | **connected to ISR to get its reg values.** |
| ISR\_reg | **connected to ISR to get its reg values.** |
| resetedISR\_index | **the number of ISR that'd've been reset.** |
| OCW2 | **connected to the OCW2 reg to know the mode.** |
| INT\_requestAck | **the ack to reset the INT\_request.** |
| serviced\_interrupt\_index | **connected to ISR (index to set) or to IRR (index to reset) the corresponding bit.** |
| zeroLevelPriorityBit | **Which bits have the highest bit priority, changes in rotation modes.** |
| INT\_request | **connected to the control logic to fire a new interrupt, Control logic will consider it's change only.** |

## Interrupt Request Register:



Figure 8: IRR

|  |  |
| --- | --- |
| Signal | Description |
| IR0\_to\_IR7 | **Input: Interrupt requests from IR0 to IR7.** |
| bitToMask | **Input: Masking bits from IMR for corresponding IRs.** |
| readPriority | **Input: Read priority signal from control logic.** |
| readIRR | **Input: Signal to output IRR values to data buffer.** |
| resetIRR | **Input: Signal from priority resolver to reset serviced interrupts.** |
| ICW1 | **Input: Initialization Command Word 1 with LTIM bit.** |
| dataBuffer | **Output: Buffer for interrupts reset by resetIRR.** |
| readPriorityAck | **Output: flag used to get read acknowledge** |
| risedBits | **Output: Rised bits indicating valid interrupts.** |

## Control Logic:

## 

Figure 9: Control Logic

|  |  |
| --- | --- |
| Signal | Description |
| INTA | **Input: Interrupt acknowledge from CPU.** |
| INT\_request | **Input: Interrupt request from priority resolver.** |
| read\_priority\_ACK | **Input: Acknowledge to deactivate the read\_priority flag.** |
| interrupt\_index | **Input: ID of the interrupt to be handled (comes from ISR).** |
| send\_vector\_ISR\_ACK | **Input: Acknowledge to deactivate the send\_vector\_ISR flag.** |
| read\_cmd\_to\_ctrl\_logic | **Input: Sent from read/write logic to read status of ISR or IRR.** |
| OCW3 | **Input: Will be used to know which register to read its status.** |
| write\_flag | **Input: to indicate that there are writing operation.** |
| ICW3 | **Input: Used in cascade mode.** |
| read\_cmd\_imr\_to\_ctrl\_logic | **Input: Read command to control logic to active IMR to read its state.** |
| ICW1 | **Input: Used to determine if operating in cascade mode or single mode.** |
| cascade\_flag | **Input: In case of slave, it will be sent from cascade controller in case that it's the desired slave** |
| SP | **Input: to determine the pic is master or slave.** |
| cascade\_signal\_ACK | **Input: Acknowledge for cascade signal from cascade controller to reset the signal.** |
| EOI | **Input: End of Interrupt signal.** |
| INT | **Output: Interrupt request will be sent to CPU.** |
| read\_IRR | **Output: Signal to read IRR status (sent to IRR).** |
| read\_priority | **Output: Set after the first INTA pulse (sent to IRR and ISR).** |
| freezing | **Output: Set between two INTA pulse.** |
| INT\_request\_ACK | **Output: Acknowledge for INT\_request flag.** |
| read\_IMR | **Input: Signal to read IMR status (sent to IMR).** |
| send\_vector\_ISR | **Output: Flag to allow ISR to send its Vector** |
| read\_ISR | **Output: Signal to read ISR status ( will be sent to ISR)** |
| pulse\_ACK | **Output: Acknowledge will be sent to ISR.** |
| second\_ACK | **Output: Determine that second INTA came (sent to ISR).** |
| EOI\_to\_cascade | **Output: Signal to cascade controller to reset cascade lines(in case of cascading mode and master).** |
| cascade\_signal | **Output: Signal to cascade controller to start working (Master mode).** |
| desired\_slave | **Output: Slave ID that will be sent to cascade controller (Master mode).** |
| cascade\_flag\_ACK | **Output: Acknowledge for cascade flag (slave mode).** |

# Testing Strategy:

We have used two strategies:

1. Unit Testing
   1. **Functionality Testing:** Test each module's basic functionality to ensure it performs its intended operations correctly.
   2. **Boundary Testing:** Test the module's behavior at its input and output boundaries. Verify how the module handles extreme values.
   3. **Error Handling Testing:** Evaluate the module's response to error conditions.
2. System Level Testing
   1. **Functional System Testing:** Test the entire PIC system's functionality to ensure that all modules work together seamlessly. This can include testing the PIC's ability to perform specific tasks or execute a set of instructions.

# Test Benches Snapshots:

A computer screen shot of a black and red line

Description automatically generated

Figure 10: Read\Write Logic TB

A screenshot of a computer

Description automatically generatedFigure 11: Priority Resolver Auto Rotation TB

A screenshot of a computer

Description automatically generated

Figure 12: Priority Resolver Fixed Priorities TB

A computer screen shot of a black and green line

Description automatically generatedFigure 13: Interrupt Request Register TB

A computer screen shot of a black screen

Description automatically generated

Figure 14: In Service Register TB

A computer screen shot of a black screen

Description automatically generated

Figure 15: Interrupt Mask Register TB

A screenshot of a computer

Description automatically generated

Figure 16: Cascade Controller TB

A computer screen shot of a black screen

Description automatically generated

Figure 17: Data Bus Buffer TB

# Team Tasks:

|  |  |  |
| --- | --- | --- |
| Name | ID | Task |
| Ahmed Khaled Abdelmaksod Ebrahim | 2000218 |  |
| Adham Khaled Abd El Maqsoud | 2000066 |  |
| Karim Ibrahim Saad Abd-Elrazek | 2001118 |  |
| Eslam Mohamed Marzouk Abdelaziz | 2000252 |  |
| Mahmoud Abdelraouf Mahmoud | 2001436 |  |
| Ahmed Mohammed Bakr Ahmed | 2000037 |  |

# References:

# GitHub Repo:

<https://github.com/Mahmoud-Abdelraouf/Computer-Organization-and-Architecture>