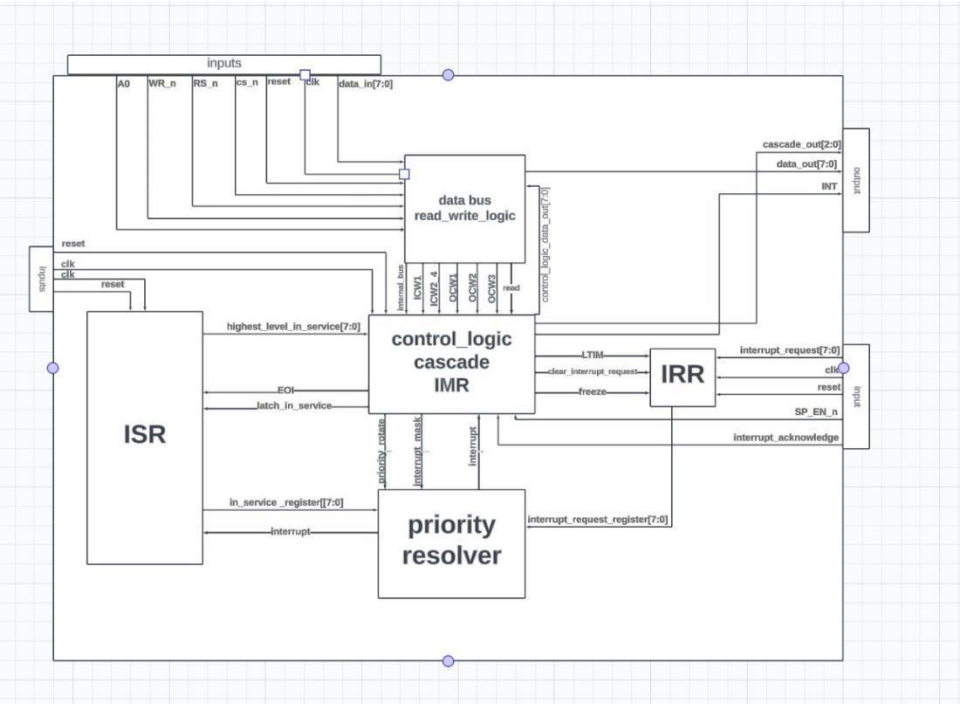
PIC Project Fall 2023

Name	ID
Abdelrahman Elsayed Ahmed Mohamed	2002139
Marwan Wael Mahmoud Abbas	2001244
Ali Mahmoud Abdulhalim Mohamed Omran	2000469
Mohamed Ayman Mohamed Solaiman	2001048
Zeyad Waleed Amin Hamaad	2000447
Omar Muhammad Mahmoud Al-Tahan	2001277

GITHUP LINK : https://github.com/D3cipherd/8259a-PIC-Project/tree/main

Table of Content:

1) Block Diagram of Design:



2) Signals' Description:

Signal Name	Description
clk	Clock signal.
reset	Reset signal.
cascade_in	Input from the cascade connection.
SP_EN_n	Signal indicating whether the interrupt controller is operating in single or cascade mode.
INTA_n	Interrupt acknowledge input.
internal_bus	Internal data bus.
ICW_1	Initialization Command Word 1 signal.
ICW_2_4	Initialization Command Word 2 to 4 signal.
OCW_1, OCW_2, OCW_3	Operation Command Words signals.
read	Read signal.
highest_level_in_service	Priority levels of interrupts being serviced.
cascade_out[2:0]	Output to the cascade connection.
cascade_io	Cascade input/output signal.
INT	Interrupt output signal.
control_logic_out_flag	Indicates whether the control logic has valid output.
control_logic_data_out	Data output from the control logic.
LTIM	Level trigger Interrupt signal.
EN_RD_REG	Enable Read Register.

read_register_isr_or_irr	Read register indicating whether ISR or IRR is being read.
interrupt_mask[7:0]	Interrupt mask for each interrupt level.
EOI[7:0]	End of Interrupt command.
freeze	Freezes the being handled interrupt.
clear_interrupt_request	Signal to clear interrupt requests.
interrupt_vector_address	Address for interrupt vectors.
ADI, SNGL, IC4	Bits of Initialization Command Word 1.
cascade_device_config	Configuration for cascade devices.
AEOI_config	Auto EOI configuration bit.
auto_rotate_mode	Auto rotate mode bit.
ack_interrupt	Acknowledged interrupts.

cascade_slave, cascade_slave_enable, cascade_out_ack2	Cascade control signals.
cmd_state[1:0], next_cmd_state	Command state variables.
ctrl_state[2:0], next_ctrl_state	Control state variables.
prev_INTA_n, nedge_interrupt_acknowledge, pedge_interrupt_acknowledge	Signals for interrupt acknowledge edge detection.
prev_read_signal, nedge_read_signal	Signals for read signal edge detection.
end_of_ack_sequence	Signal indicating the end of the acknowledge sequence.
interrupt_from_slave_device	Signal indicating interrupts from cascade slave devices.
interrupt_when_ack1	Interrupts when in ACK1 state.
CS_n	Chip Select signal (active low).

RD_n	Read signal (active low).
WR_n	Write signal (active low).
A0	Address bit 0.
data_in[7:0]	Input data bus.
internal_bus[7:0]	Internal data bus.
stable_A0	Stable version of the address bit 0.
read	Read signal (output control).
level_or_edge_triggered_mode	Input from control logic indicating trigger mode (level or edge).
clear_ir_line[7:0]	Input from control logic to clear individual interrupt lines.
ir_req_pin[7:0]	Input from I/O devices representing individual interrupt request lines.
interrupt_req_reg[7:0]	Output representing the interrupt request register.
priority_rotate[2:0]	Input specifying the priority rotation.
interrupt[7:0]	Input representing individual interrupt lines.
latch_in_service	Input indicating whether to latch interrupts into the in-service register.
end_of_interrupt[7:0]	Input specifying individual end-ofinterrupt signals.
in_service_register[7:0]	Output representing the in-service register.
interrupt_req_reg[7:0]	Input representing individual interrupt request lines.
masked_interrupt_req_reg[7:0]	Wire representing the masked interrupt request register.

masked_in_service_register[7:0]	Wire representing the masked inservice register.
rotated_interrupt_request[7:0]	Wire representing the rotated interrupt request lines.
rotated_interrupt[7:0]	Wire representing the rotated and resolved interrupt lines.
rotated_in_service[7:0]	Wire representing the rotated inservice register.
in_service_priority_mask[7:0]	Register representing the priority mask based on in-service register.

3) Brief Description of Testing Strategy:

1) Module Instantiation:

- The testbench instantiates a PIC module using input/output signals from and to CPU.
- The PIC module is connected to various control signals and data buses.

2) Clock Generation:

- The clock (**clk**) is generated with a specified period (CLK_PERIOD) using an initial block with a forever loop.

3) Stimulus/Test Case Generation:

- The initial block initializes various signals to their default values.
- It then generates stimulus to test different functionalities of the PIC module.
- The test sequence includes initialization, ICW1 (Initialization Control Word 1), ICW2, ICW3, ICW4, OCW1 (Operational Control Word 1), OCW2, OCW3, handling interrupts, sending acknowledgment, and non-specific EOI (End of Interrupt).

4) Interrupt

- Interrupts are simulated by setting the **interrupt_request** signal to specific values.

- The testbench simulates both single-bit interrupts and interrupts with a combination of bits.

5) Acknowledgment Handling:

- The testbench simulates the acknowledgment process by toggling the **interrupt_acknowledge_n** signal.

6) End of Interrupt Handling:

- The testbench simulates the end of interrupt by sending a nonspecific EOI command to the PIC module or Initialized Automatic EOI in OCW2.

7) Cascade Mode Testing:

- The testbench tests the cascade mode of the PIC module by configuring ICW3 accordingly.
- It also tests the cascading process and acknowledgment in the cascade mode.

8) Multiple Test Cycles:

- The testbench repeats similar test sequences in multiple cycles, covering different configurations and scenarios.

9) Simulation Termination:

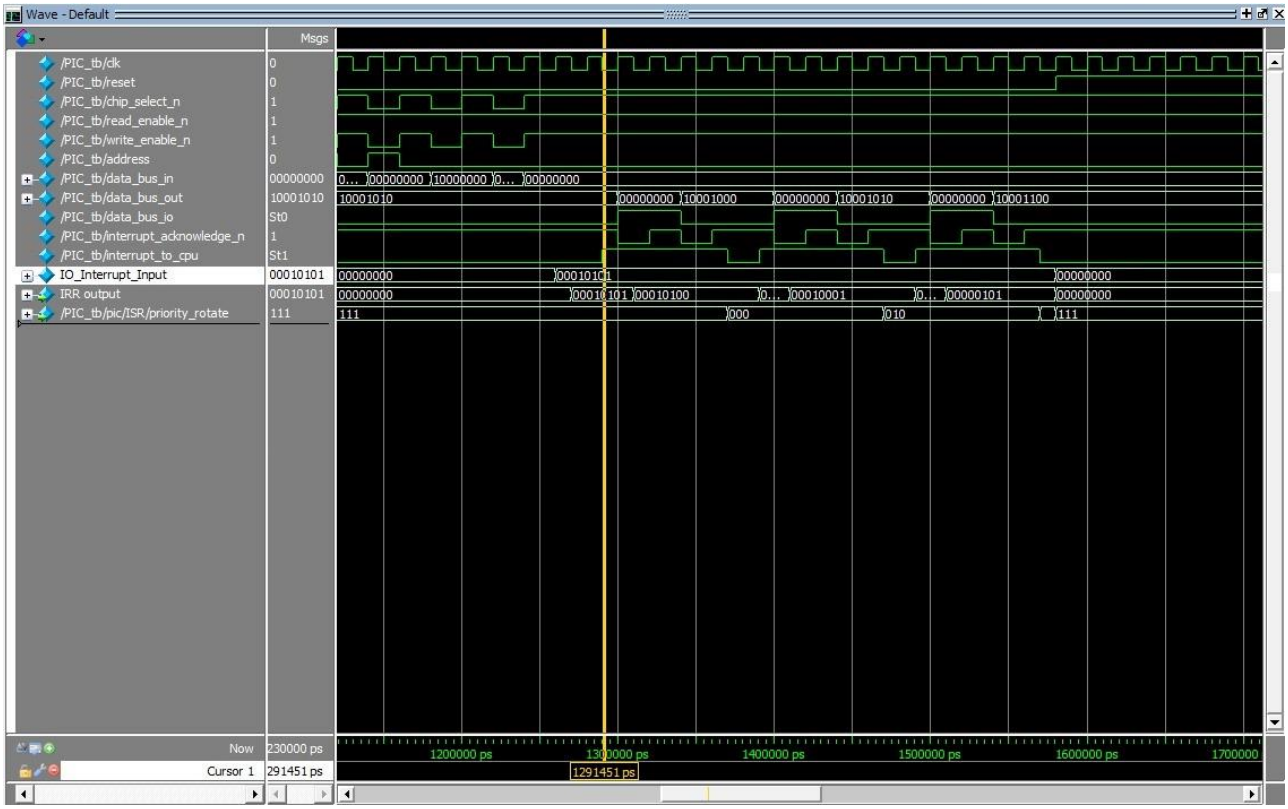
- The simulation is terminated after a specific number of cycles using the **$stop** system task.

10) Comments and Delays:

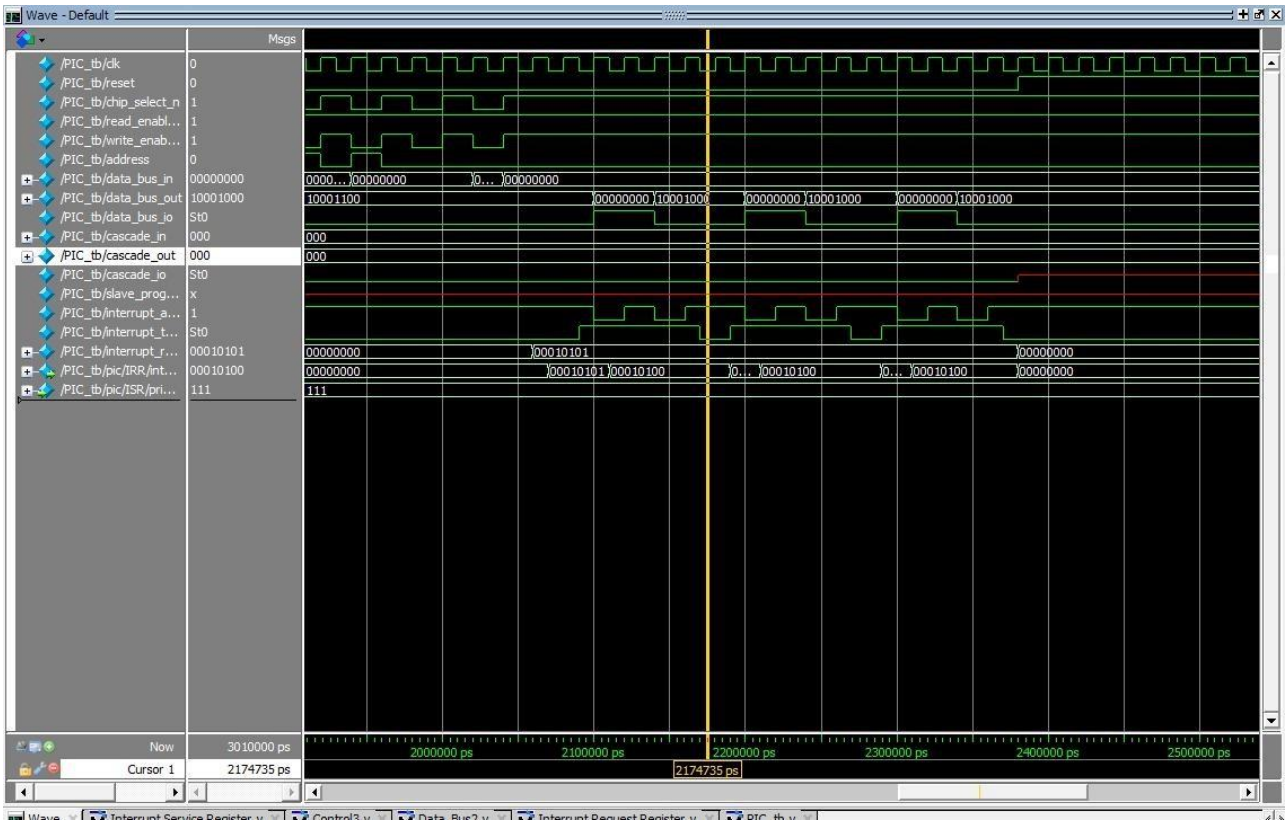
- The testbench includes comments to describe the purpose of each block.
- Delays (**#**) are used to control the timing of events in the simulation.

4) Snapshots of Simulation Waves:

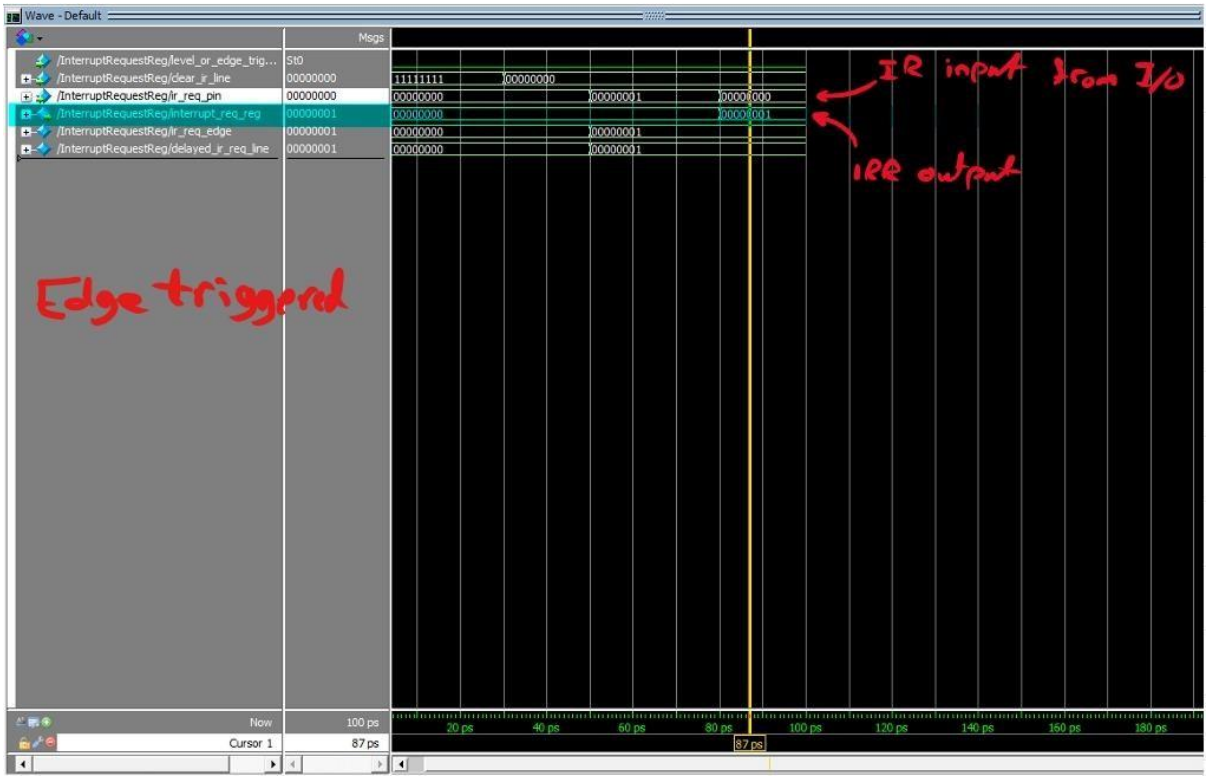
Different Testcases/Modes of 8259a PIC



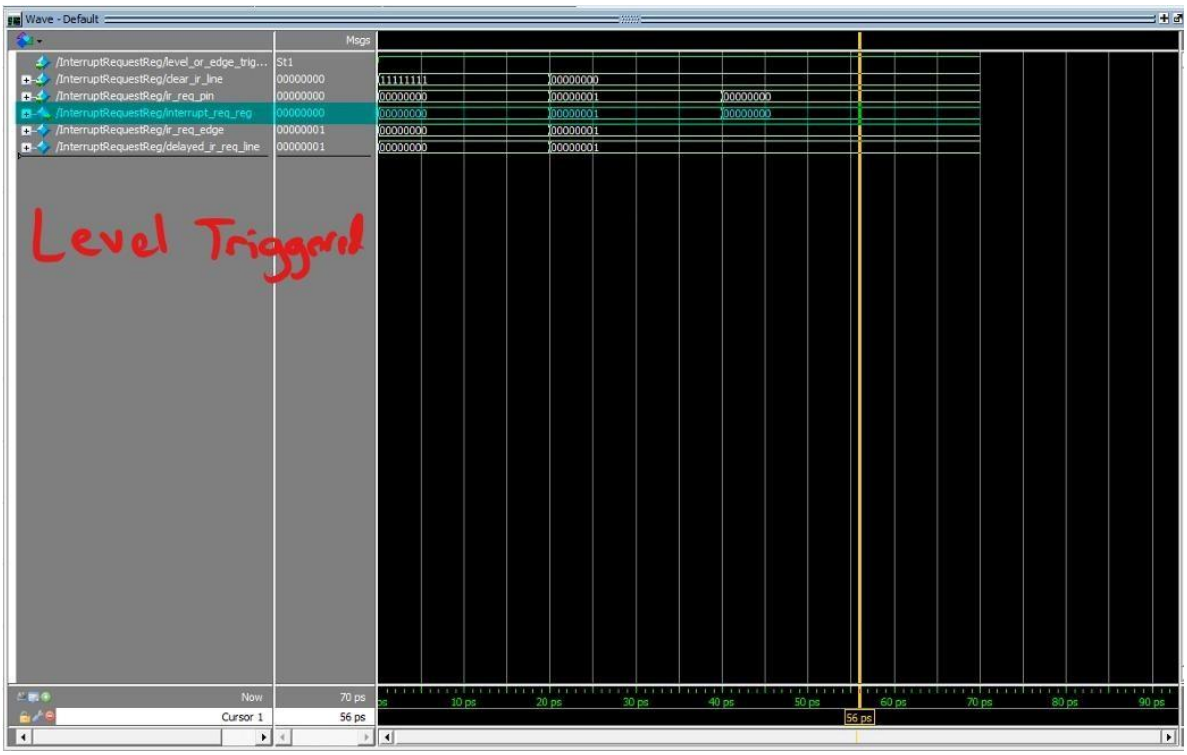
AEOI with Rotation



AEOI without Rotation



Edge Triggered



Level Triggered

5) Team Members' Contributions:

Name	CONTRIBUTION
Abdelrahman Elsayed Ahmed Mohamed	IRR, ISR & Testbench
Marwan Wael Mahmoud Abbas	Control logic & Testbench
Ali Mahmoud Abdulhalim Mohamed Omran	Control logic & Testbench
Mohamed Ayman Mohamed Solaiman	Data bus & Read Write & REPORT
Zeyad Waleed Amin Hamaad	Priority Resolver & REPORT
Omar Muhammad Mahmoud Al-Tahan	CASCADE MODE IN CONTROL LOGIC MODULE & REPORT

6) Additional Information:

1- WE merge some modules with each other to reduce the number of modules to:

1. **Simplified Architecture:** **Merging modules can simplify the overall architecture of the DMA controller by reducing the number of individual functional units. This simplification can make the controller easier to understand, maintain, and debug.**

2. **Reduced Hardware Complexity:** **By combining multiple modules or functionalities into a single module, there can be a reduction in hardware complexity. This may lead to more efficient use of resources and potentially lower manufacturing costs.**

3. **Improved Performance:** **In some cases, merging modules can lead to improved performance. By integrating certain functions, it might be possible to reduce latency, streamline data pathways, or optimize processes, resulting in better overall performance of the DMA controller.**

4. **Enhanced Flexibility and Adaptability:** **A more consolidated module might offer greater flexibility and adaptability. It can potentially handle a wider range of tasks or operations by virtue of being a more versatile unit, thereby improving the controller's overall functionality.**

5. **Power and Space Efficiency:** **Integrating modules can sometimes lead to more efficient use of power and physical space on the integrated circuit. This is particularly beneficial in embedded systems or devices where power consumption and size constraints are critical factors.**

2- we add clock from cpu to dma module to make the module :

1. **Synchronization: The clock signal synchronizes the operation of different parts of the system. In the case of DMA, it ensures that data transfers between memory and peripherals occur at the correct timing and in a coordinated manner. The DMA controller relies on the clock signal to perform its operations in sync with other system components.**

2. **Control of Operations:** **The clock signal acts as a fundamental timing reference for controlling the operation of the DMA controller. It helps in regulating when the DMA**

controller fetches data, performs transfers, and completes operations, ensuring that these actions occur in a predictable and coordinated manner.

3. **Data Transfer Timing:** The clock signal defines the timing for transferring data between different parts of the system, such as between memory and I/O devices. The DMA controller uses the clock signal to time its requests and responses for data transfer, ensuring data integrity and reliability.

4. **Power Management:** In some systems, the clock signal might be used for power management purposes. Clock gating techniques can be employed to control when the DMA controller is active or in a low-power state, helping to conserve energy when the DMA functionality is not required.

3- we add reset to dma module to make the module:

1. **Initialization:** DMA reset signals are used to initialize the DMA controller. When the system boots up or when there's a need to reinitialize the DMA controller, the reset signal helps set the DMA to a known state, ensuring that it starts its operation correctly.

2. **Clearing State:** It helps in clearing the internal state of the DMA controller. This is particularly useful to ensure that any previous configurations or data left over from a previous operation are cleared, avoiding any potential conflicts or errors in subsequent DMA transfers.

3. **Error Handling:** Reset signals are instrumental in error handling and recovery. When the DMA controller encounters an unexpected error or becomes unresponsive, a reset signal can be sent to bring the DMA controller back to a known state, allowing for a fresh start and potential recovery from the error condition.

4. **Synchronization:** In some cases, a reset signal can be used for synchronization purposes. For example, ensuring that all involved components are in sync by resetting them simultaneously before starting a coordinated operation involving DMA transfers.

5. **System Stability:** It contributes to the stability of the system by providing a means to reset and recover DMA functionality in case of unexpected behavior, ensuring the proper functioning of data transfers without causing system instability.