Team 26:

1. **1-** Andrew Ayman Samir 2000003
2. **2-** Mazen Essam Eldin Helmy 2001607
3. **3-** George Geham William 2000073
4. **4-** Sara Ahmed Mostafa 2000362
5. **5-** Youssef Saad Gobran 2001440
6. **6-** Philopeteer Sameh Rasmy 2000955

8259 PIC Project Report

CSE311s Computer Architecture

Contents

GitHub Repo:

[YoussefGobran/PIC-8259-Arch (github.com)]

Contribution Table:

Name	Contribution
Andrew Ayman	**PIC main control logic module & Cascading Testbench**
Mazen Essam Eldin	**Initialization and Configuration module**
George Geham	**Single Mode AEOI Edge Triggered Testbench**
Sara Ahmed	**Priority Resolver Handler Module**
Youssef Saad	**Cascading Handler Module & Single Mode EOI Level Triggered Testbench**
Philopeteer Sameh	**Reading Status Handler Module**

Project Description:

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.

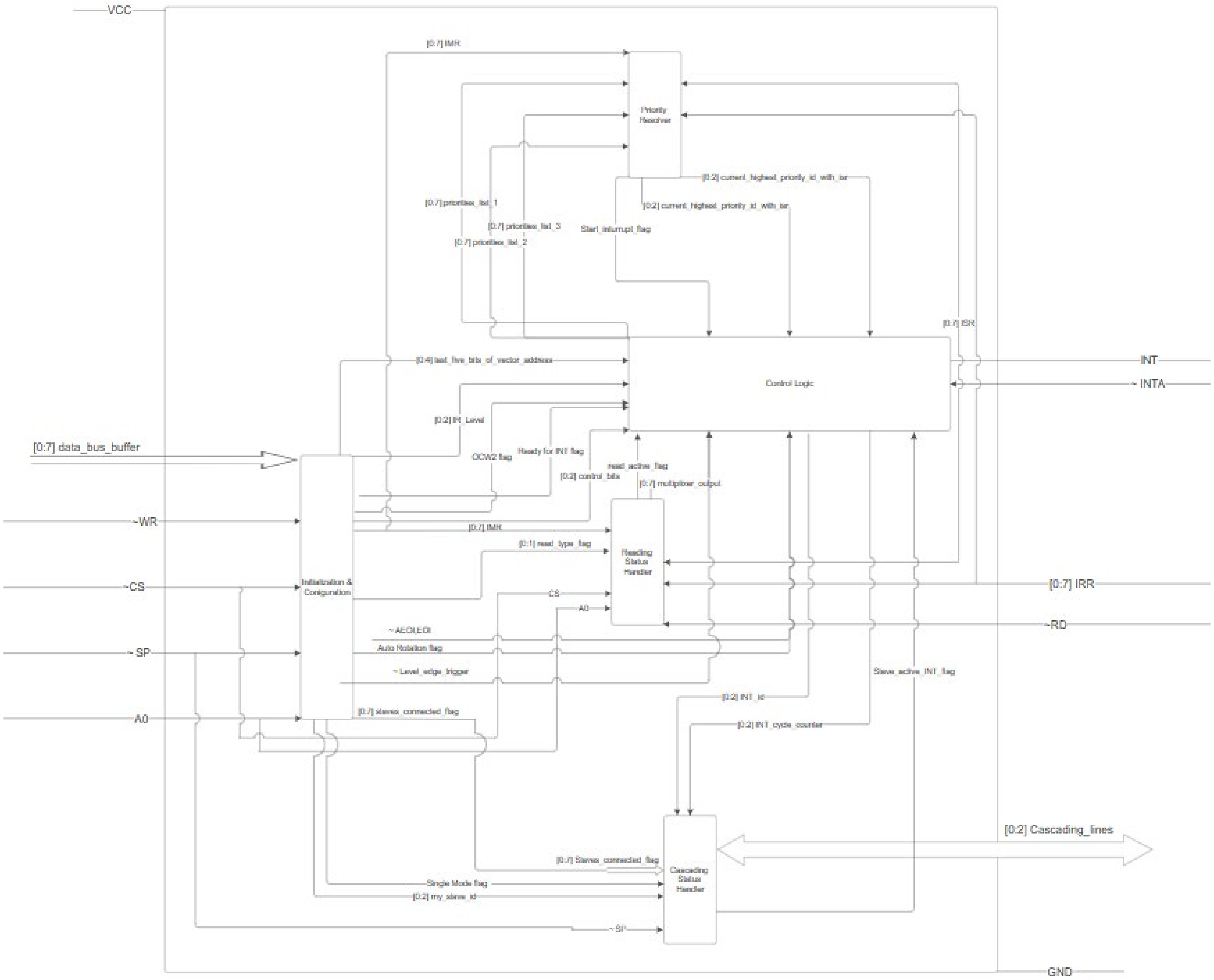
Functionalities Provided:
[8259 Compatibility, Programmability (ICWs,OCWs), Cascade Mode, Interrupt Handling, Interrupt Masking through IMR, Edge/Level Triggering, Fully Nested Mode, Automatic Rotation, EOI, AEOI, Specific Rotation, Reading Status]

Description for all signals used

Signal	Description
cs_neg	used to activate the reading and writing protocols(active_low).
rd_neg	used to read the status of the pic(irr,isr,imr) (active_low).
wr_neg	used to write ICWs, OCWs to PIC to initalize and configure some important flags(active_low).
a0	used with the data bus buffer in reading and writing protocols.
sp_eng	used in cascade mode to flag the module as slave (active_low).
inta_neg	used to receive the interrupt acknowledged pulses during the interrupt cycle. (The first to initalize the isr, The second to release the vector address onto the data bus buffer).
vcc	giving 5V+ power to pic (Not used in Verilog as it has no meaning from simulation perspective).
gnd	giving GND power to pic (Not used in Verilog as it has no meaning from simulation perspective).
ir[0:7]	the interrupt lines coming from the I/O devices (Slave Interrupt Lines) which is used to flag to PIC what interrupt is active.
data_inout[0:7]	The data bus buffer used in: Initialization, Configuration, reading, releasing vector address.
cas[0:2]	the cascading lines which is used as a chip select to the active slave with the interrupt.
int	The interrupt line used to flag to the CPU that there is an active interrupt.
interrupt_cycle_counter	used internally to check where exactly is the pic during the interrupt cycle (Think of it as an FSM).
data_bus_buffer_output[0:7]	used to control what values it has when releasing the vector address.
output_data_bus_control	to control when the data_out line have the data bus buffer output.
interrupt_active	to check internally if there is an active interrupt at all times.
interrupt_active_id[0:2]	the id of the corresponding higher prioirty interrupt after the first inta pulse which this id will be set in the isr.

isr[0:7]	holds 1 in the corresponding active id after the first inta pulse in the interrupt cycle.
irr[0:7]	holds 1 in the corresponding id after initialization from ir and resets the value in edge triggering mode after setting of the isr after the first inta pulse int the interupt cycle.
single_mode_flag **level_trigger_flag_and_edge_level_neg** **last_five_bits_of_vector_address[0:4]** **slaves_connected_flag[0:7]** **my_slave_id[0:2]** **aeoi_and_eoi_neg_flag** **imr[0:7]** **ir_level_ocw2[0:2]** **control_bits_ocw2[0:2]** **ocw2_output_flag** **automatic_rotation_mode_flag** **read_type_flag[0:1]** **ready_to_accept_interrupts_flag**	flags that are calculate from the ICW 1 ,2,3,4, OCW 1,2,3 which are needed in the rest of the design in conditional checks along the code.
read_active_flag	to flag that there is a read output should be done unto the data_out.
read_data_buffer	what the value the data_out when read_active_flag is active.
[0:2] priorities[0:7] **priorities_list_1[0:7]** **priorities_list_2[0:7]** **priorities_list_3[0:7]**	used as a prioirty controller (8x3) where each row as and ir level and the value of the row is the prioirty level where initially ir0 has a priority of 7 and so forth.
current_highest_priority_id[0:2]	the id of the corresponding higher prioirty interrupt which is calculated from the masked_irr.
current_highest_priority_id_with_isr[0:2]	the id of the corresponding higher prioirty interrupt which is calculated from the masked_irr with the isr used in cascading mode to release the cascading line.
interrupt_start_flag	to start the interrupt cycle and increase the interrupt cycle.

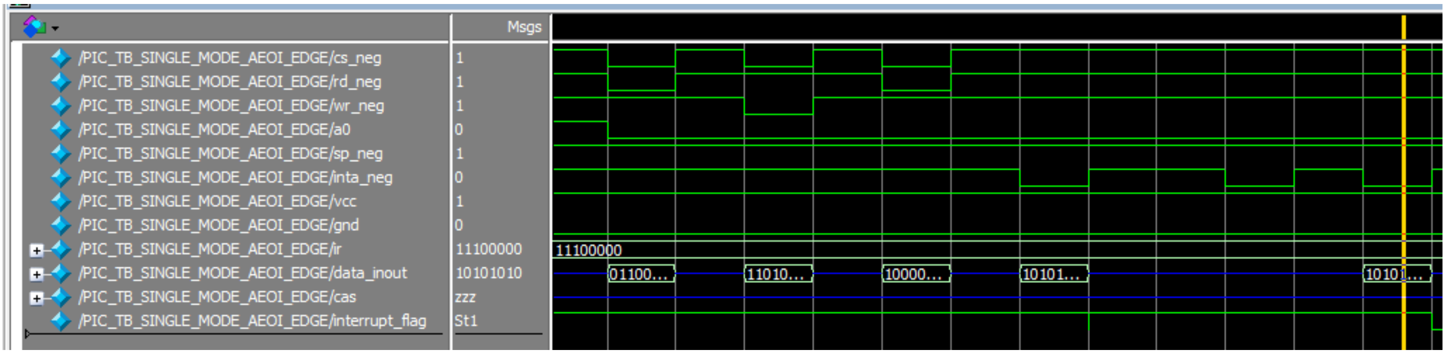
Block Diagram:



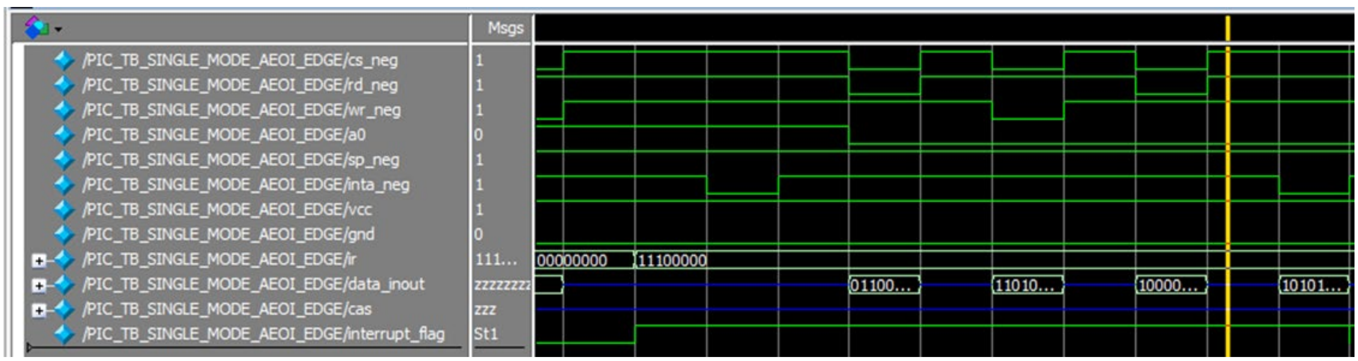
Brief description of the testing strategy

We decided to divide the testbench into three different test benches.
1. Single PIC module with AEOI and set to edge Triggering features tested are: Programmability, Interrupt Priority Handling, Masking, Edge Triggering, Fully Nested Mode, Automatic Rotation in AEOI, AEOI Functionality and Reading status of the pic during runtime.
2. Single PIC module with EOI and set to level triggering, features tested are: Programmability, Interrupt Priority Handling, level triggering, automatic rotation in EOI , EOI functionality, specific rotation handling.
3. multiple PIC module with AEOI and set to edge triggering, we connected two slave modules to master module, features tested are: Cascading with right handling of operations, Programmability, Interrupt Priority Handling, Edge triggering, AEOI,.

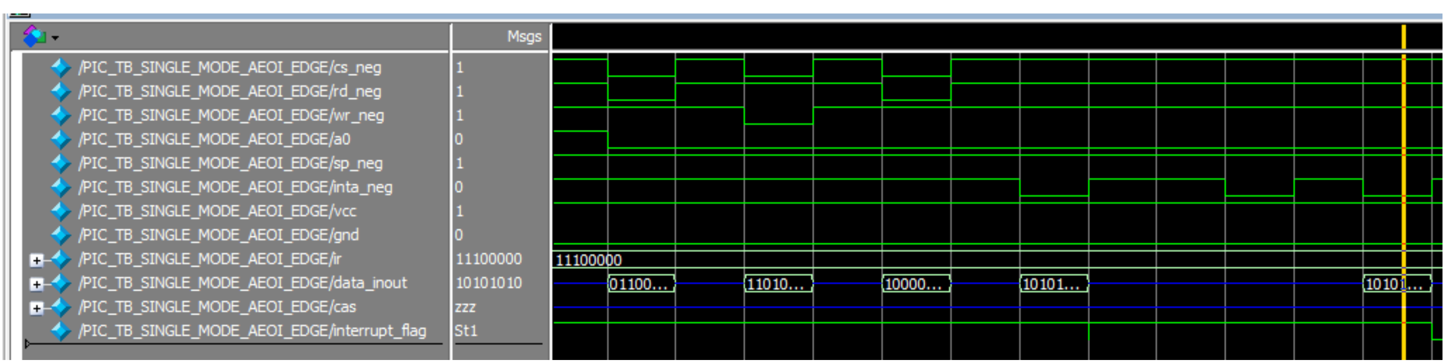
1. First Testbench



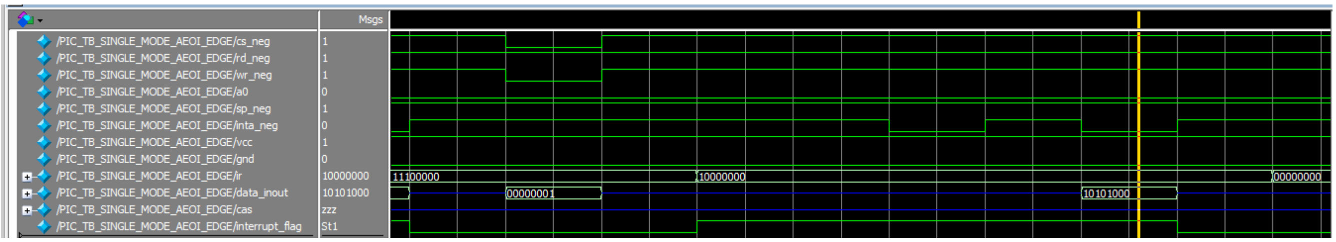
Initialize the PICs through ICW1,ICW2,ICW4 then setting of the IMR disabling IR1 through OCW 1.

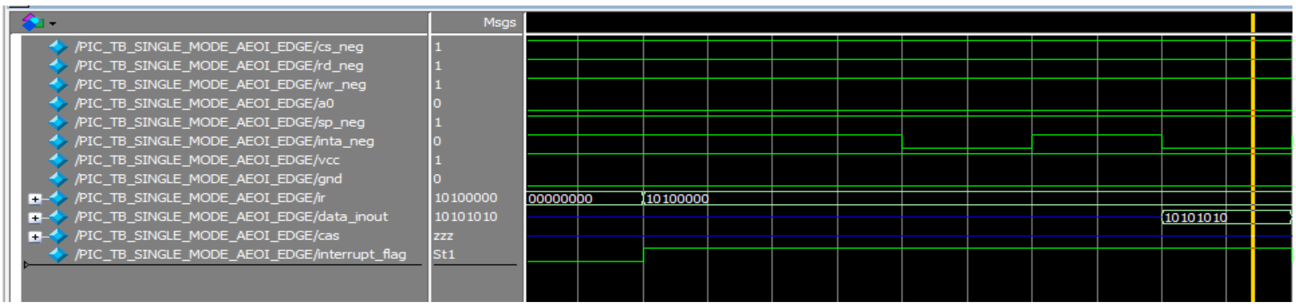


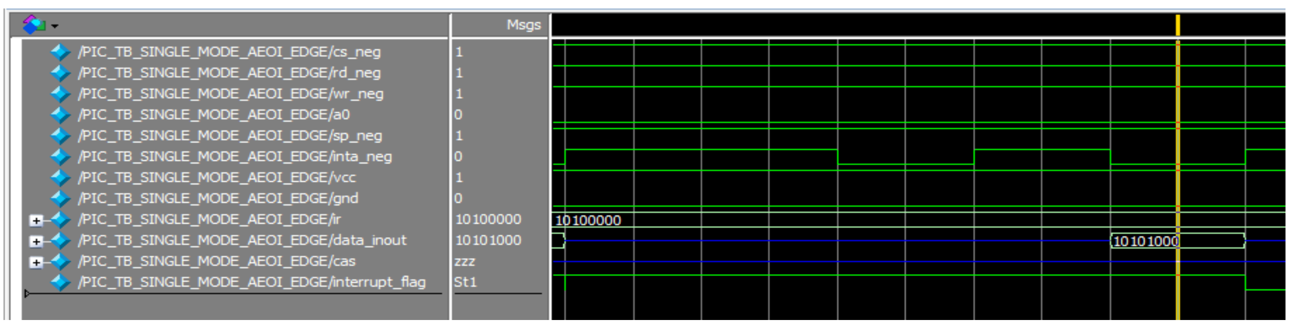
IR0-2 is active, and the priority chooses IR0, we send the first INTA pulse to set the ISR then we read IRR which gives the correct value that IRR0 is disabled and the rest as they are. Then we need to read the isr but first send an ocw3 to make the next reading pulse read the isr , sending OCW3 then reading isr which will have the coresponfing bit to IR0 as active, and finally send the second pulse during the data bus buffer has the vector address of the IR0 then the pulse ends to close the interrupt cycle (AEOI).



When we set the IMR, we disabled IR1 so the next interrupt would be of IR2 not 1 as it will appear when the second INTA pulse the vector address of IR2 is on the data bus buffer.

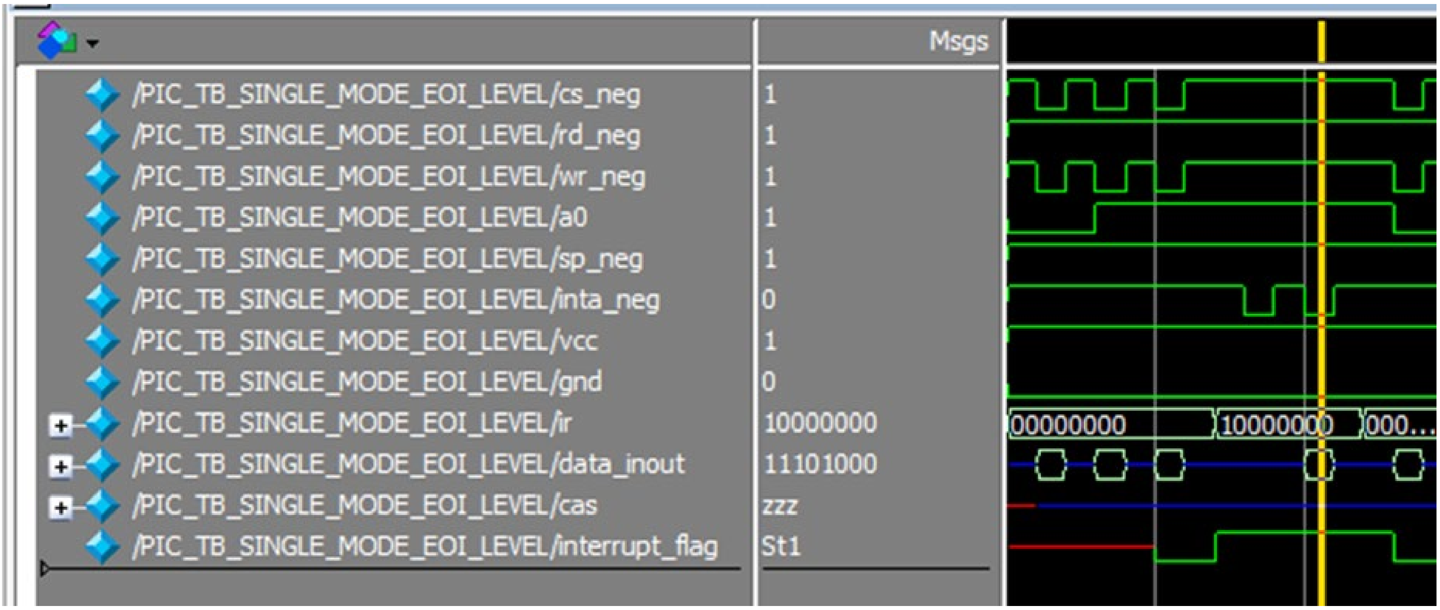




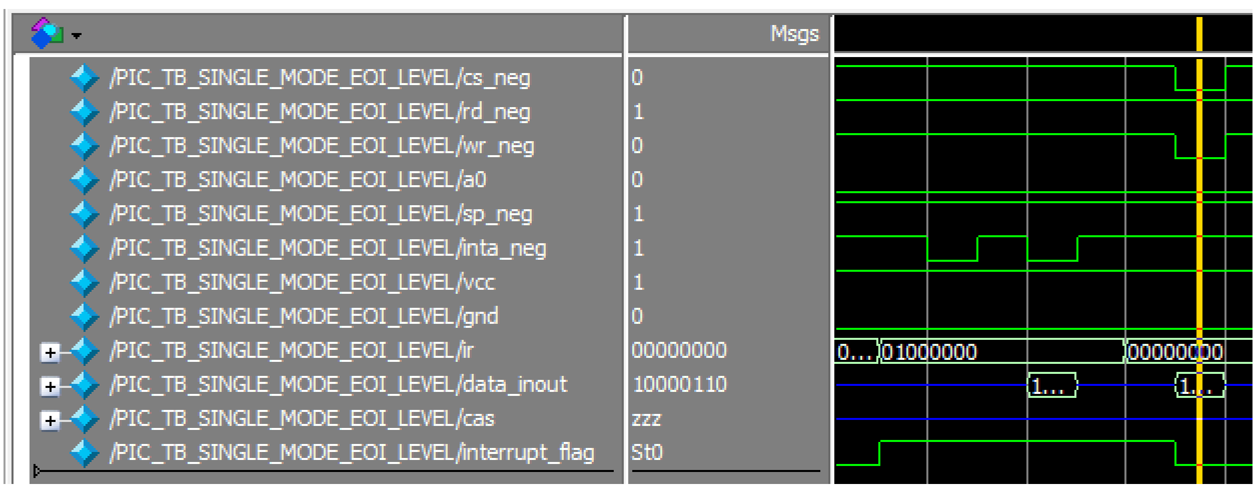


Then we send an OCW2 to set automatic rotation then we make IR0 and the send the corresponding INTA pulses which at the end because of automatic rotation The IR0 becomes the least priority, then we make the ir0,2 active again which because of the auto rotation earlier will first choose IR2 then IR0.

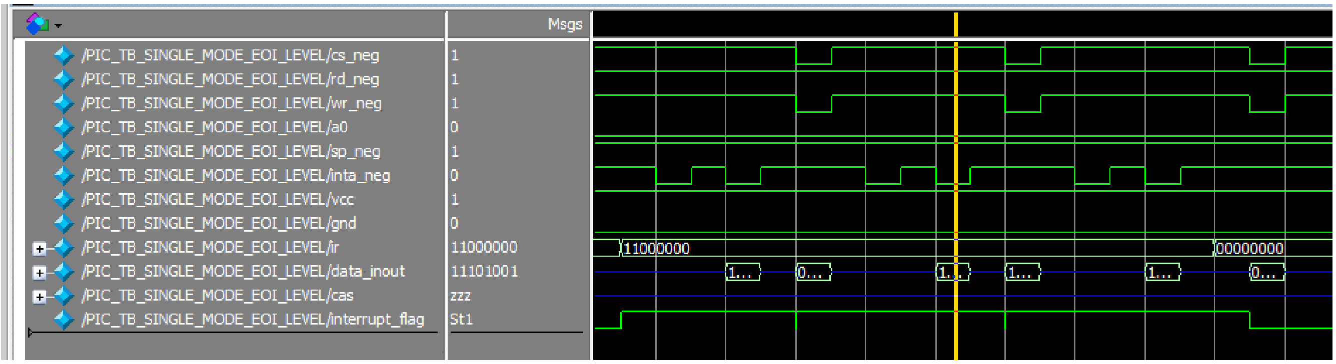
2. Second Testbench



Initialize the PICs through ICW1,ICW2,ICW4 then setting of the IR0 making the interrupt flag as on then sending the 2 pulses of INTA but the interrupt flag wouldn't come down at the end of the second pulse because of it needs to receive an EOI first so this time we send non-specific EOI through ocw2 to end the interrupt cycle, we set ir0 back to 0 to not restart the cycle because of level triggering.



Test IR0 again with specific EOI command but don't forget to disable ir0 before the end because of level triggering.



We set IR0,1 and send two pulses of INTA pulses then send non-specific rotate EOI to reset ir0.

Then it chooses ir1 because of rotated ir0 then we send specific rotate on IR1 to reset it then it will choose IR0 back again because of rotated ir1, notice we didn't have to reset IR during all those three interrupts because of Level Triggering .

3. Third Testbench

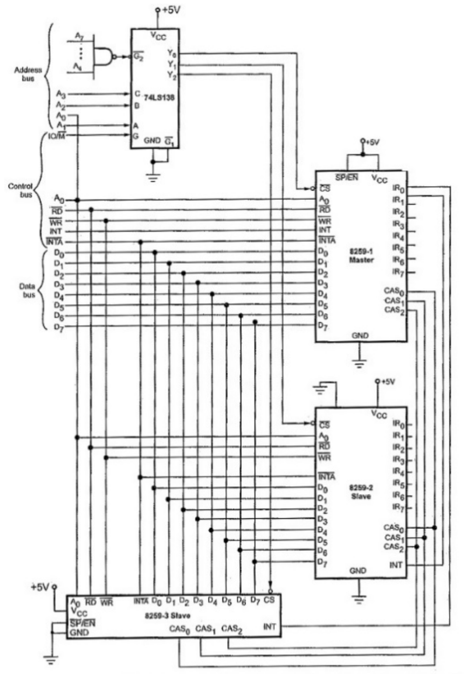
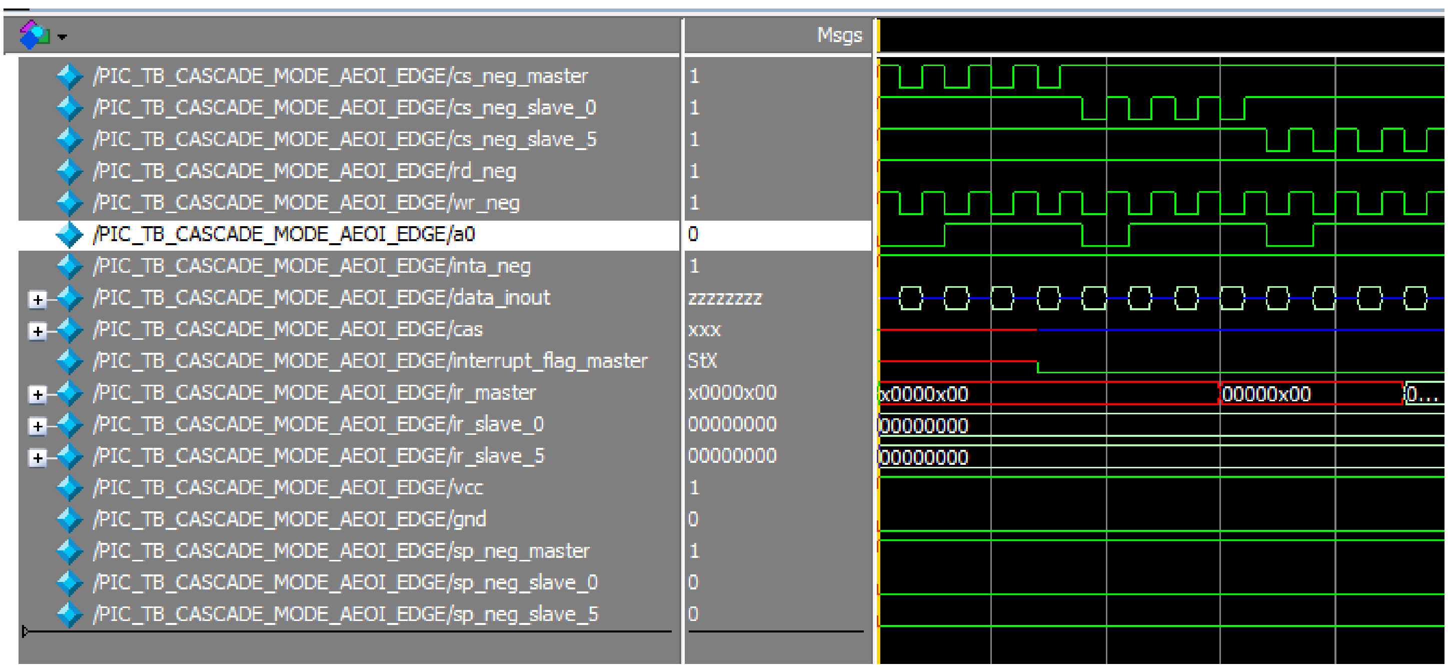
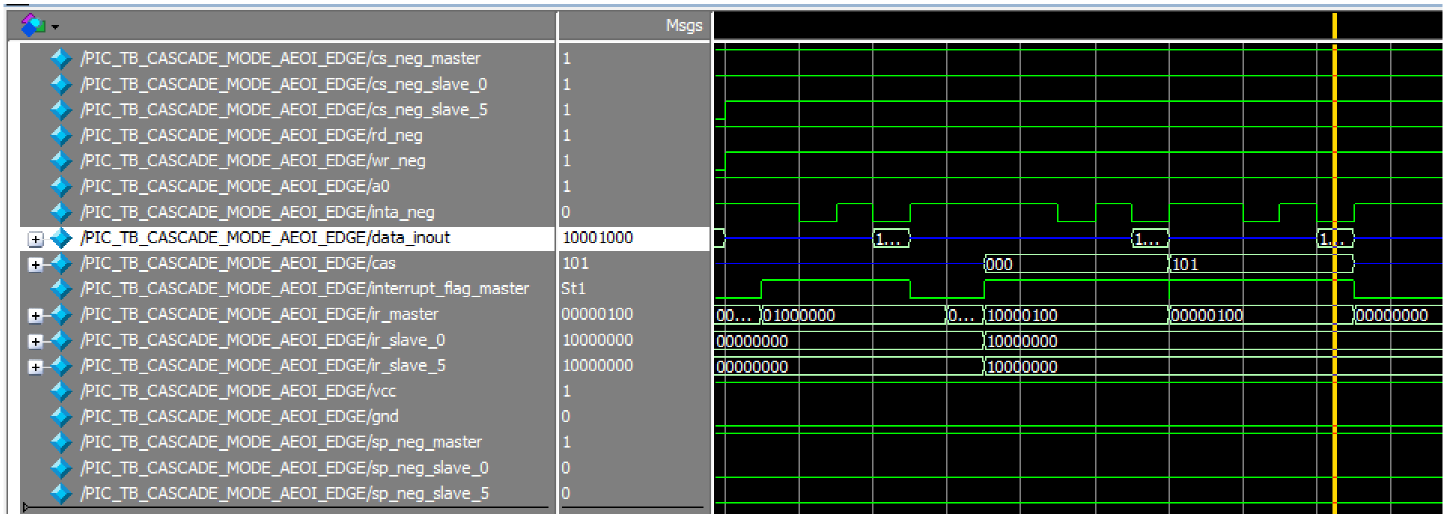


Fig. 14.74 Three 8259 in the cascade mode

A schema like this was used but instead of slaves connected to IR0,IR1 it instead to IR0,IR5



We set the three pics with the appropriate ICW1,2,3,4 to all three pics so a total of 12 words with the appropriate cs used.



We first tested ir1-master which is not connected to any slave to make sure that the normal interrupts are handled correctly, we second test setting of IR0-slave-0, IR0-slave-5 which will make IR0,5 -master active send the appropriate pulses which will notice the cascading lines the appropriate id of the slave connected to the active IR. After the first two pulses the cycle of the IR0-master is finished then it will choose IR5-master and send the appropriate two pulses and the correct vector address is shown (the last five bits set from the ICW2, and the corresponding id of IR).
Finally, it finished the cycle, and the interrupt flag is down with the end of the second pulse because of AEOI mode.

Thank You