

PIC 8259A Project

Computer Architecture

A rectangular black and white lines with numbers

Description automatically generated with medium confidence

From Intel’s Data Sheet

|  |  |
| --- | --- |
| Name | ID |
| Abdallah Mohamed | 2001803 |
| Ahmad Youssef | 2002238 |
| Mohamed Mostafa | 2001299 |

GitHub link

https://github.com/AntiHexCode/PIC-8259A.git

# Table of Contents

### Overview

* Sequence of operation
* Block diagrams
* Signals
* Simulation
* Test bench methodology
* modifications

# Overview

This project simulates 8259A PIC behavior using Verilog, PIC is short for **P**rogrammable **I**nterrupt **C**ontroller. The design was inspired from the [Intel datasheet](https://drive.google.com/file/d/1ff_bdktK6zrH54DNJ6saOl5jdVz1-0MY/view?usp=drive_link) with some modifications.

The design was divided into 4 major blocks as follows:

* Control logic block
* Interrupt logic block
* Read Write logic block
* Cascade logic block

Our l PIC 8259A is designed to be:

* 8086 compatible
* Programmable
* Single +5V supply, no master clock
* Eight-Level Priority Controller
* Expandable to 64 Levels via cascading
* Handling interrupts in fully nested mode/automatic rotation
* Interrupt masking compatible
* EOI/AEOI supportive
* supportive for reading status.

# sequence of operation

1. All command words are sent from 8086 to the RW logic.
2. RW logic parses the command words sending flags to control logic
3. Whilst command words are being sent, all blocks are initializing according to the command words
4. Once all command words are sent, other blocks can start working on the interrupt.
5. Control logic triggers 8086 for interrupts
6. Interrupt starts upon receiving the first INTA(active low) pulse, fetching the IRs
7. Priority resolver chooses which request will be served taking into consideration various modes(fully-nested, rotation mode etc...)
8. Control logic puts the vector address(from ISR) on the data bus upon receiving the second INTA pulse only if address Write flag is high (in single mode), in case of cascade mode, depending on current interrupt location, it would be put on the data bus by one of the slaves.
9. 8086 sends read signal, allowing to read ISR(current interrupt request in service), IRR and IMR

# Control Logic Block

## Block Diagram

## 

A blue screen with white text

Description automatically generated

## Block Ports

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Pin Name | Pin Type | In | Bits | What it indicates |
| A0 | Input |  | 1 | Used to decipher various commands (ICWs and OCWs) |
| wren | Input |  | 1 | When high then CPU writes commands (ICWs or OCWs) to the PIC on the Data Bus (D7-D0) |
| rden | Input |  | 1 | when high then CPU reads status (IRR, ISR or IMR) from the PIC On the Data Bus (D7-D0) |
| DBus | Input |  | 8 | The ICWs, OCWs, PIC Status or Vector Address get transferred via the data bus |
| ICW1flag | Input |  | 1 | When high then the CPU writes ICW1 to the PIC on the Data Bus (D7-D0) |
| ICW2flag | Input |  | 1 | When high then the CPU writes ICW2 to the PIC on the Data Bus (D7-D0) |
| ICW3flag | Input |  | 1 | When high then the CPU writes ICW3 to the PIC on the Data Bus (D7-D0) |
| ICW4flag | Input |  | 1 | When high then the CPU writes ICW4 to the PIC on the Data Bus (D7-D0) |
| OCW1flag | Input |  | 1 | When high then the CPU writes OCW1 to the PIC on the Data Bus (D7-D0) |
| OCW2flag | Input |  | 1 | When high then the CPU writes OCW2 to the PIC on the Data Bus (D7-D0) |
| interruptLocation | Input |  | 3 | The location of the highest priority interrupt to the IRR |
| LTIM | Output | ICW1 | 1 | Determines if the PIC works in the level (when high) or edge interrupt mode |
| ADI | Output | ICW1 | 1 | The Address Interval |
| SNGL | Output | ICW1 | 1 | When high then there’s no cascading nor ICW3 command issued |
| IC4 | Output | ICW1 | 1 | When high then ICW4 command must be read from the CPU |
| TReg |  | ICW2 | 5 | Adding these 5 bits to the 3 bits of the interrupt location to get the vector address |
| SReg | Output | ICW3 | 8 | Determines which IR input has a slave (when high) and which has not |
| SFNM | Output | ICW4 | 1 | When high then the PIC works in the special fully nested mode |
| BUF | Output | ICW4 | 1 | Determines if the PIC in buffered mode or not with the help of MS flag |
| MS | Output | ICW4 | 1 | Determines if the PIC in buffered mode or not with the help of BUF flag |
| AEOI | Output | ICW4 | 1 | When high then the End of Interrupt (EOI) will be Automatic |
| MReg | Output | OCW1 | 8 | Interrupt Mask |
| R | Output | OCW2 | 1 | End of Interrupt, Automatic Rotation, Specific Rotation |
| SL | Output | OCW2 | 1 | End of Interrupt, Automatic Rotation, Specific Rotation |
| EOI | Output | OCW2 | 1 | End of Interrupt, Automatic Rotation, Specific Rotation |
| LReg | Output | OCW2 | 3 | IR Level to be acted upon |
| readIRR | Output |  | 1 | When high then the IRR is transferred to the CPU in the read status process |
| readISR | Output |  | 1 | When high then the ISR is transferred to the CPU in the read status process |
| readIMR | Output |  | 1 | When high then the IMR is transferred to the CPU in the read status process |
| vectorAddress | Output |  | 8 | Address of the subroutine to be executed to handle the interrupt |

## Test Bench

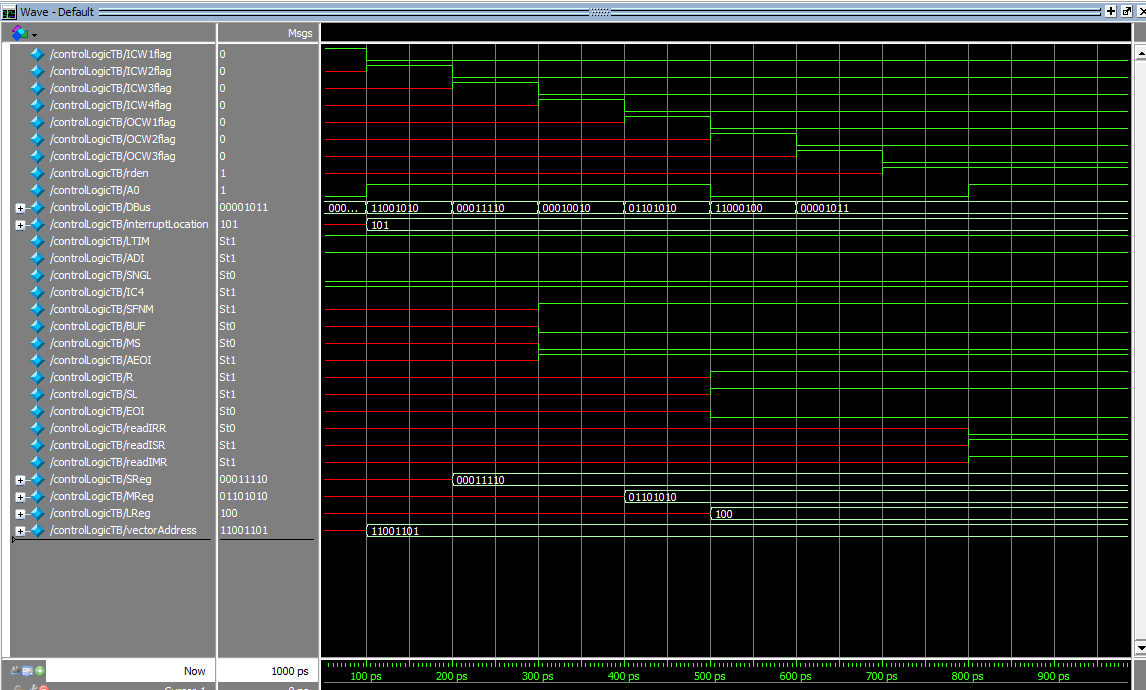


Figure 1: All ICWs and OCWs received.

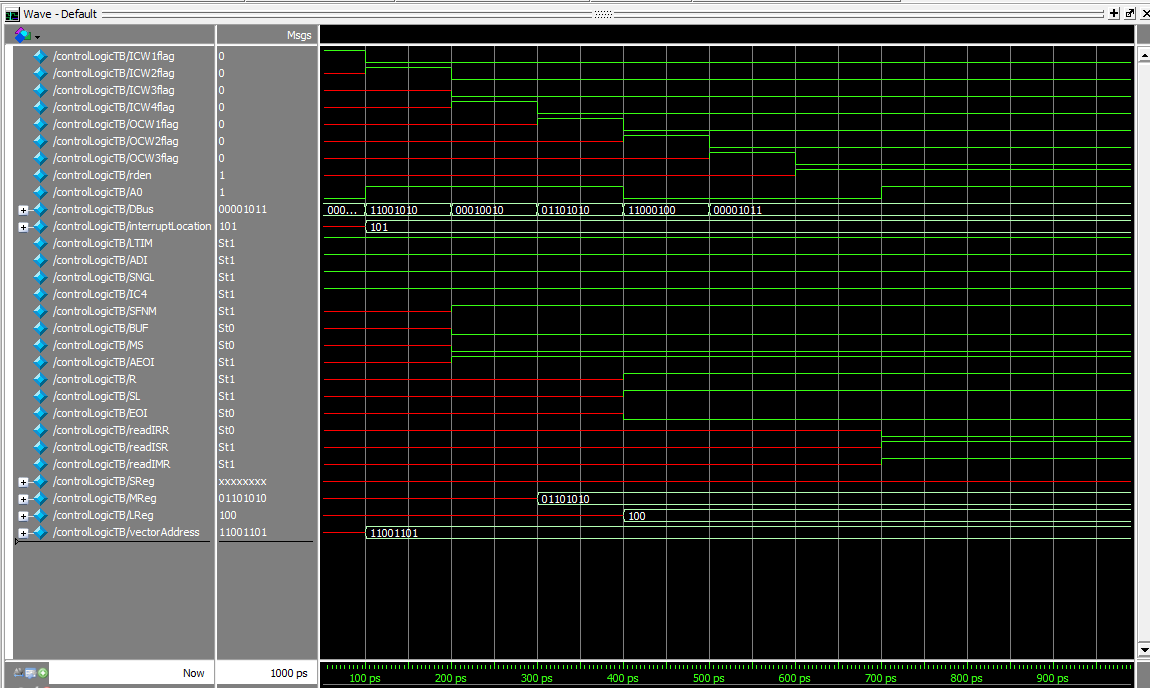


Figure 2: All ICWs (except ICW3) and All OCWs

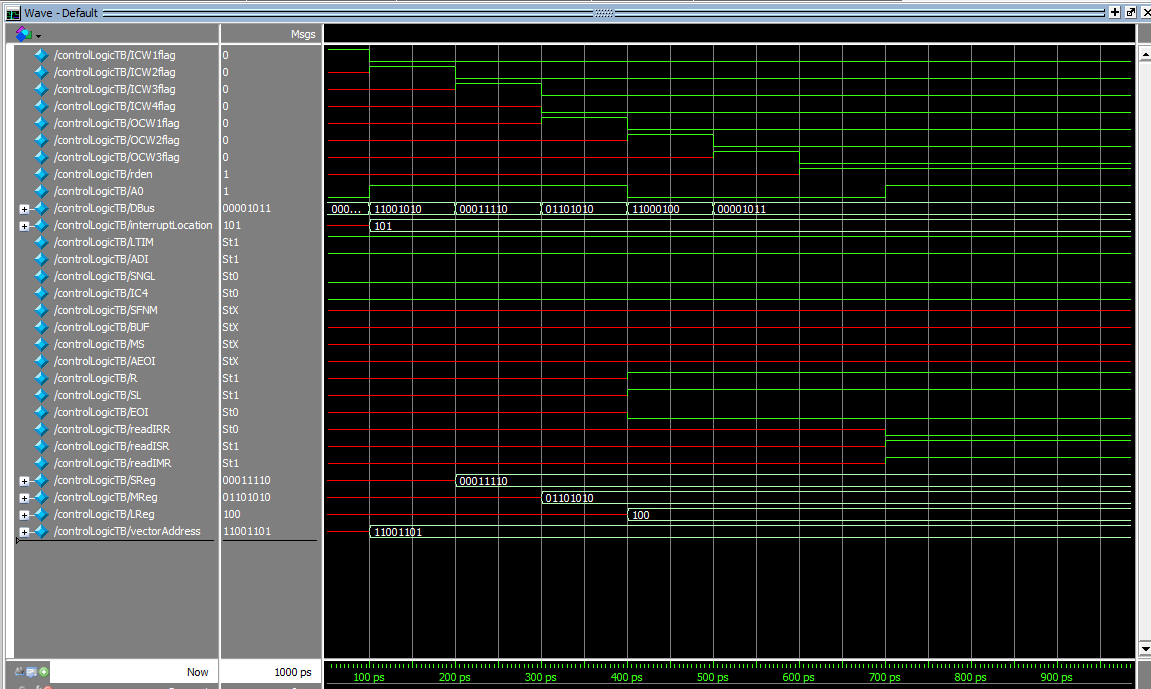


Figure 3: All ICWs (except ICW4) and all OCWs

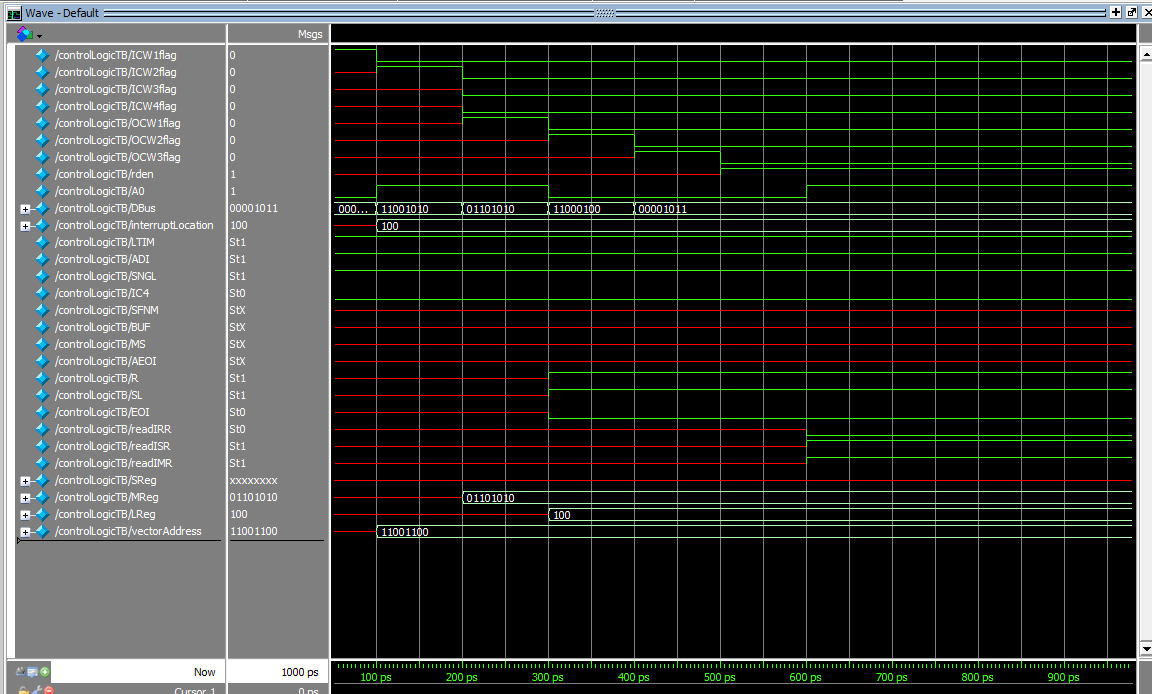


Figure 4: All ICWs (except ICW3&4) and all OCWs

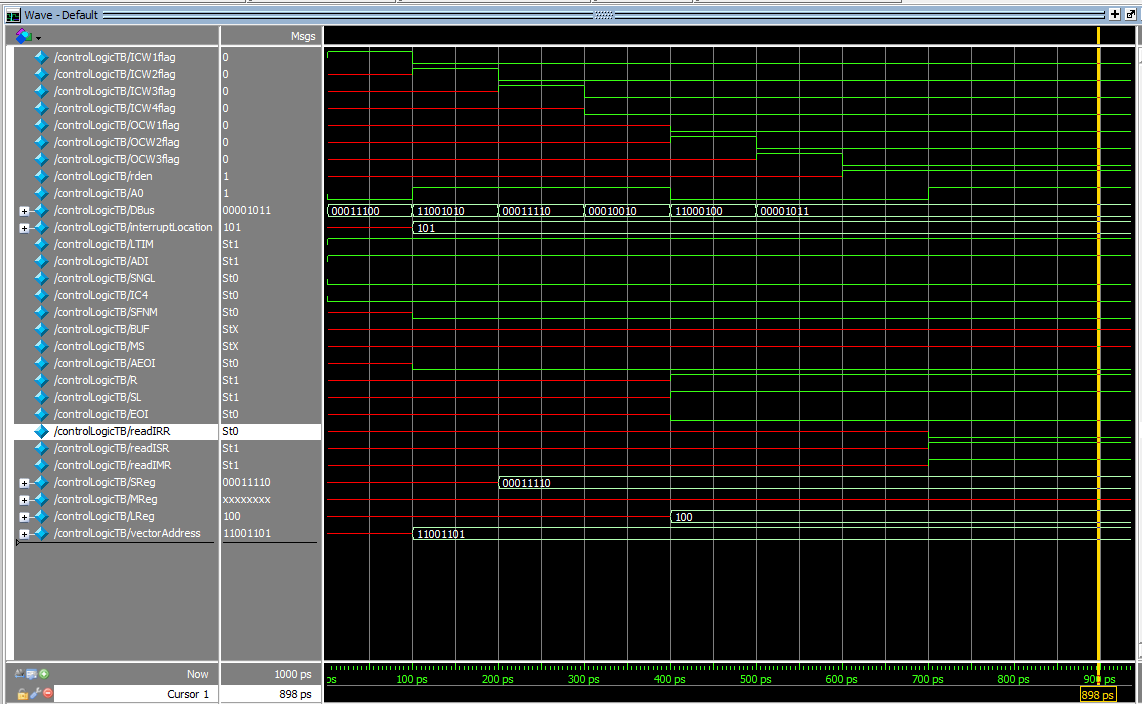


Figure 5: All ICWs (except ICW4) and all OCWs, but this case with default SFNM and AEOI values

# Read Write Logic

## Block Diagram

A white rectangle with black text

Description automatically generated

## Block Ports

|  |  |  |  |
| --- | --- | --- | --- |
| Pin Name | Pin Type | Bits | What it indicates |
| A0 | Input | 1 | Used to decipher various commands (ICWs and OCWs) |
| CS | Input | 1 | When active the CPU could read or write using the WR or RD |
| WR | Input | 1 | When active (also CS should be active) then CPU writes commands (ICWs or OCWs) to the PIC on the Data Bus (D7-D0) |
| RD | Input | 1 | When active (also CS should be active) then CPU reads status (IRR, ISR or IMR) from the PIC On the Data Bus (D7-D0) |
| dataBus | Input | 8 | The ICWs, OCWs, PIC Status or Vector Address get transferred via the data bus |
| ICW1 | Output | 1 | When high then the CPU writes ICW1 to the PIC on the Data Bus (D7-D0) |
| ICW2 | Output | 1 | When high then the CPU writes ICW2 to the PIC on the Data Bus (D7-D0) |
| ICW3 | Output | 1 | When high then the CPU writes ICW3 to the PIC on the Data Bus (D7-D0) |
| ICW4 | Output | 1 | When high then the CPU writes ICW4 to the PIC on the Data Bus (D7-D0) |
| OCW1 | Output | 1 | When high then the CPU writes OCW1 to the PIC on the Data Bus (D7-D0) |
| OCW2 | Output | 1 | When high then the CPU writes OCW2 to the PIC on the Data Bus (D7-D0) |
| OCW3 | Output | 1 | When high then the CPU writes OCW3 to the PIC on the Data Bus (D7-D0) |

# Cascade Logic

## Block Diagram

A diagram of a computer system

Description automatically generated

## Block Ports

|  |  |  |  |
| --- | --- | --- | --- |
| Pin Name | Pin Type | Bits | What it indicates |
| SP/EN | Input | 1 | If high means The PIC is master, else and it’s a slave |
| SReg | Input | 8 | stores either the 8 bits distinguishing slaves connecting to a master PIC or holds the 3-bit ID for a slave (least significant 3 bits) |
| decInterruptLocation | Input | 1 | Interrupt Location in decimal (from 0 to 7) |
| interruptLocation | Input | 3 | Interrupt Location in binary (from 000 to 111) |
| CASBus | Inout | 3 | The master sends on it the ID of the slave that has an interrupt which is going to be served |
| mPermissionToWrite | Output | 1 | When high then the master is allowed to put the vector address on the bus |
| sPermissionToWrite | Output | 1 | When high then the slave is allowed to put the vector address on the bus |

# Interrupt Logic

## Block Diagram

A diagram of a device

Description automatically generated

## Block Ports

|  |  |  |  |
| --- | --- | --- | --- |
| Pin Name | Pin Type | Bits | What it indicates |
| IRBus | Input | 8 | The devices send the requests via the IRBus |
| LTIM | Input | 1 | Level-Triggered Interrupt Mode |
| SFNM | Input | 1 | Special Fully Nested Mode |
| AR | Input | 1 | Automatic Rotation |
| AEOI | Input | 8 | Automatic End of Interrupt |
| TReg | Input | 5 | 5-bits to which we append another 3 bits to get the 8-bit subroutine address |
| IMR | Input | 1 | Interrupt Mask Register |
| readIMR | Input | 1 | A flag to indicate that the CPU wants to read IMR |
| readISR | Input | 1 | A flag to indicate that the CPU wants to read ISR |
| readIRR | Input | 1 | A flag to indicate that the CPU wants to read IRR |
| INTA# | Output | 1 | Interrupt acknowledge from the CPU to the PIC |
| decInterruptLocation | Output | 1 | Location of the interrupt in decimal (0 to 7) |
| interruptLocation | Output | 3 | Location of the interrupt in binary (000 to 111) |
| internalBus | Output | 8 | The bus connecting some modules inside the PIC |

## testbench

A screenshot of a computer

Description automatically generated

# Test bench methodology

* Initializing using command words (ICWs), testing all possible cases of missing ICW3 and ICW4, and their existence
* Writing command words
* Setting interrupt requests
* Testing fully nested mode
* Testing read status mode
* Testing AEOI

# modifications

* R/W logic works with an internal clock, since the command words need some form of sequence to operate, a clock was needed to enhance and ease the design of the logic of command words.
* All blocks won't start working unless all command words are sent.
* 8086 must send all OCWs to facilitate the design of the blocks.
* RW logic takes some of the control logic tasks such as parsing the data for command words and sends them to control logic.
* Control logic and R/W logic can be reduced to one single complex block.
* Interrupt logic block receives the acknowledgement (INTA) directly from 8086.
* Control logic sets the 8-bit vector address on the data bus not the ISR.
* Control logic is responsible for reading the status of PIC, in exchange for R/W logic parsing the data and setting flags.