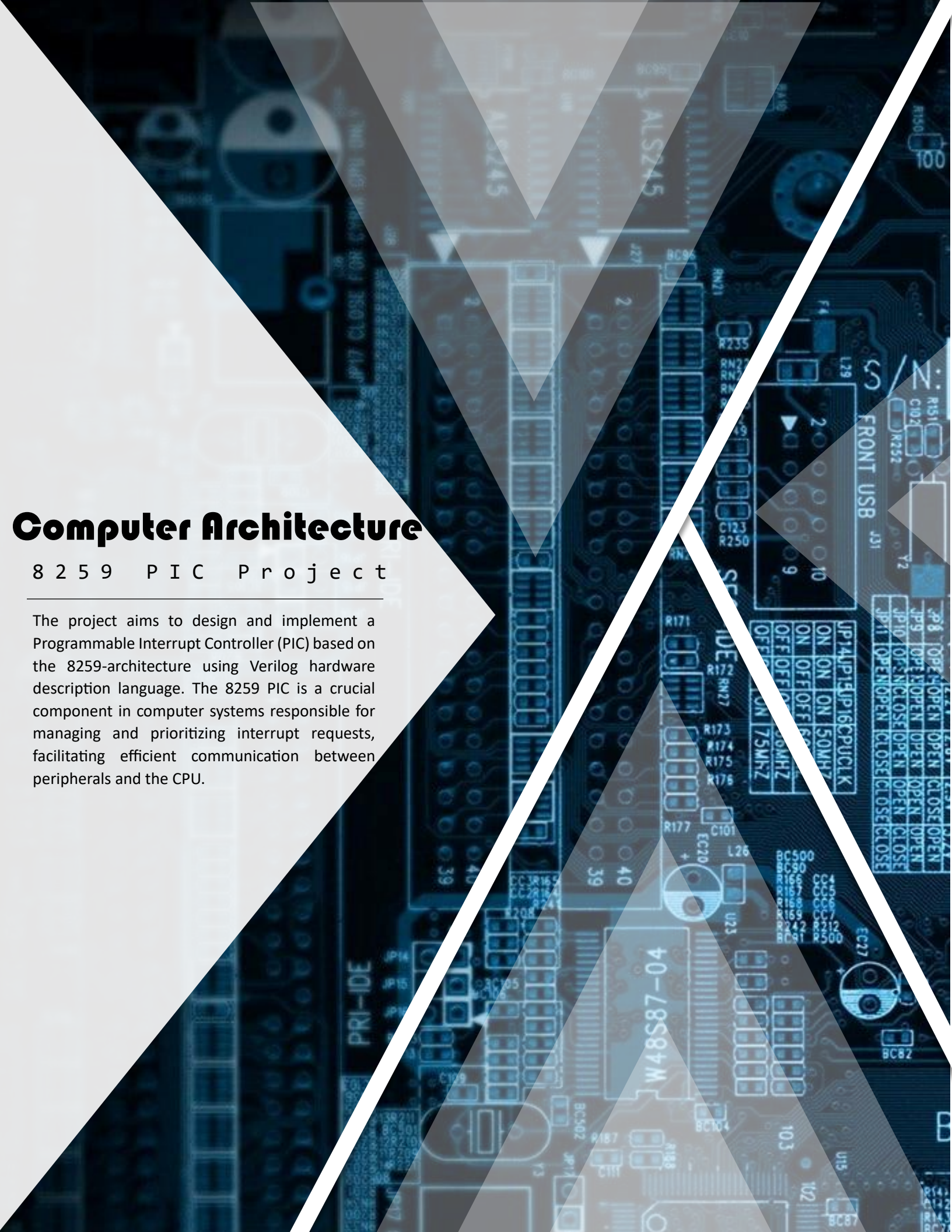

Computer Architecture

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.

1.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

2.Table Of Contents:

Table of Contents

3.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.

4.Block Diagram:

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

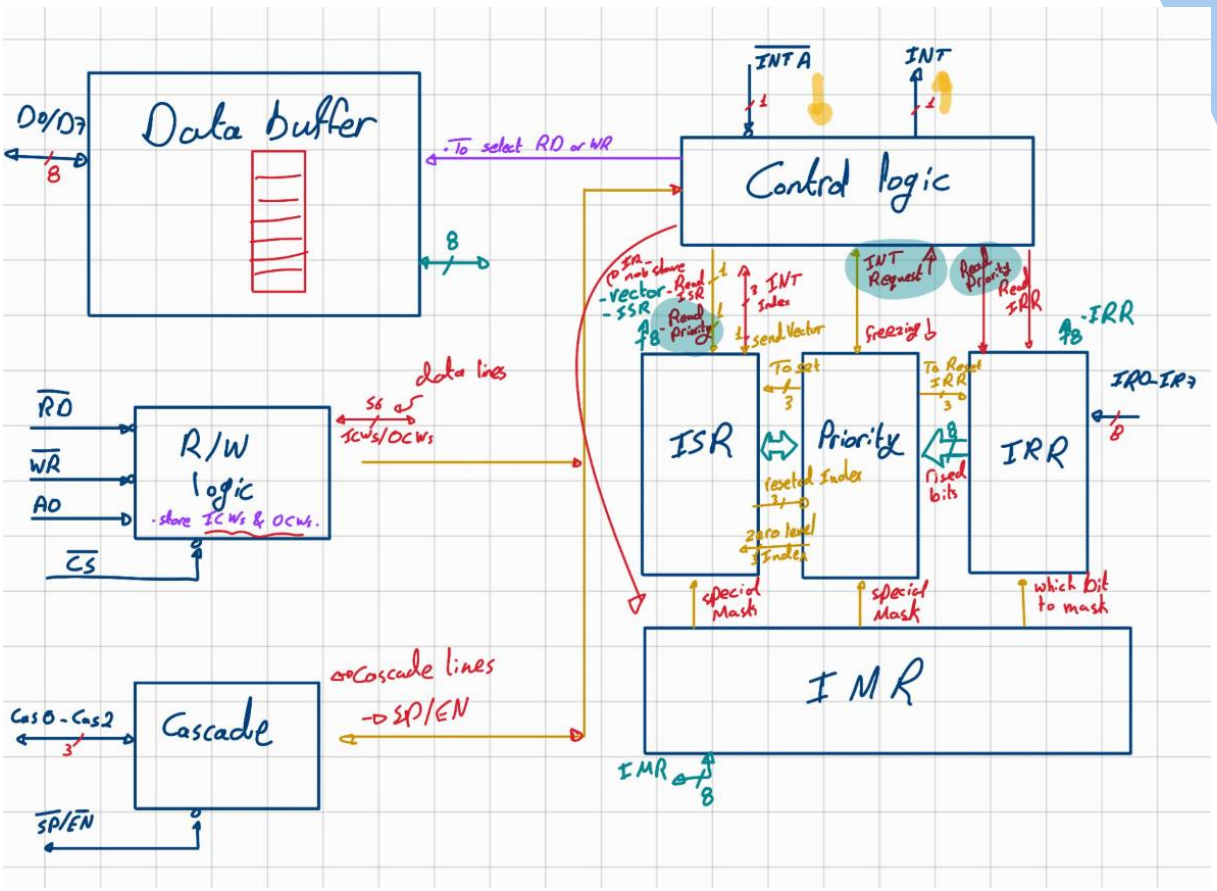

Figure 1. neat sketch for block diagram.

5.Signals Description:

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

a) Data Buffer:

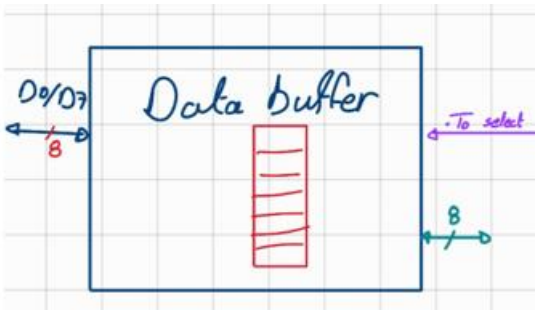

Figure 2. Data Buffer

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

b) R\W Logic:

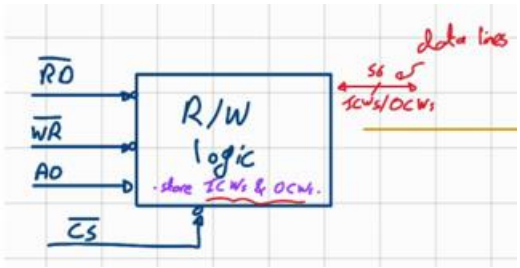

Figure 3. R\W Logic

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.

c) Cascade Controller:

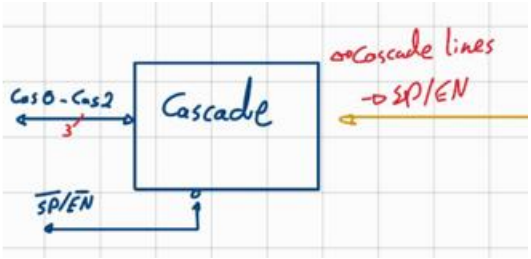

Figure 4: Cascade Controller

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)

d) IMR:

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.

e) In Service Register:

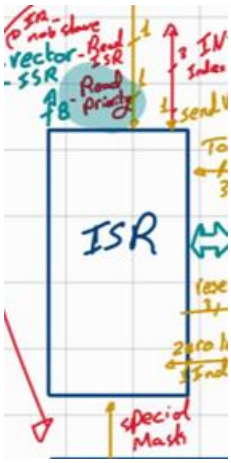

Figure 6: ISR

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

f) Priority Resolver:

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.

g) Interrupt Request Register:

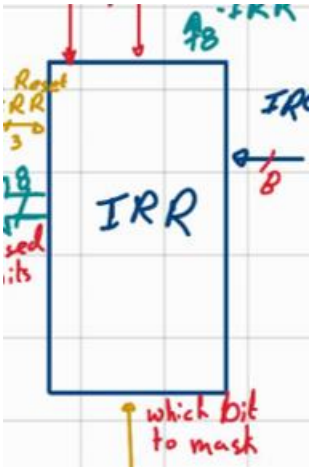

Figure 8: IRR

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.

h) Control Logic:

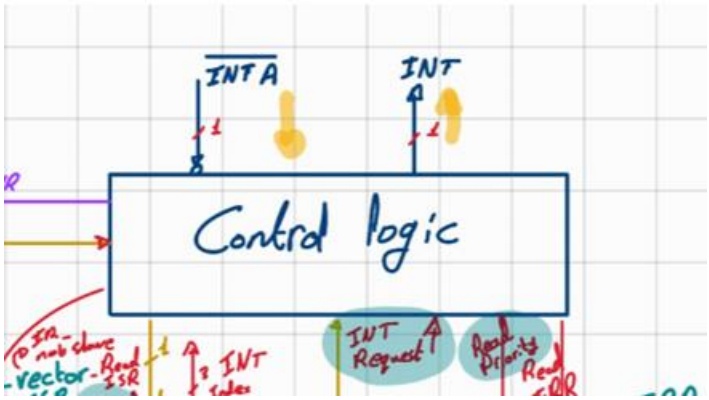

Figure 9: Control Logic

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).

6. Testing Strategy:

We have used two strategies:

1. Unit Testing
    a. **Functionality Testing:** Test each module's basic functionality to ensure it performs its intended operations correctly.
    b. **Boundary Testing:** Test the module's behavior at its input and output boundaries. Verify how the module handles extreme values.
    c. **Error Handling Testing:** Evaluate the module's response to error conditions.

2. System Level Testing
    a. **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.

7. Test Benches Snapshots:

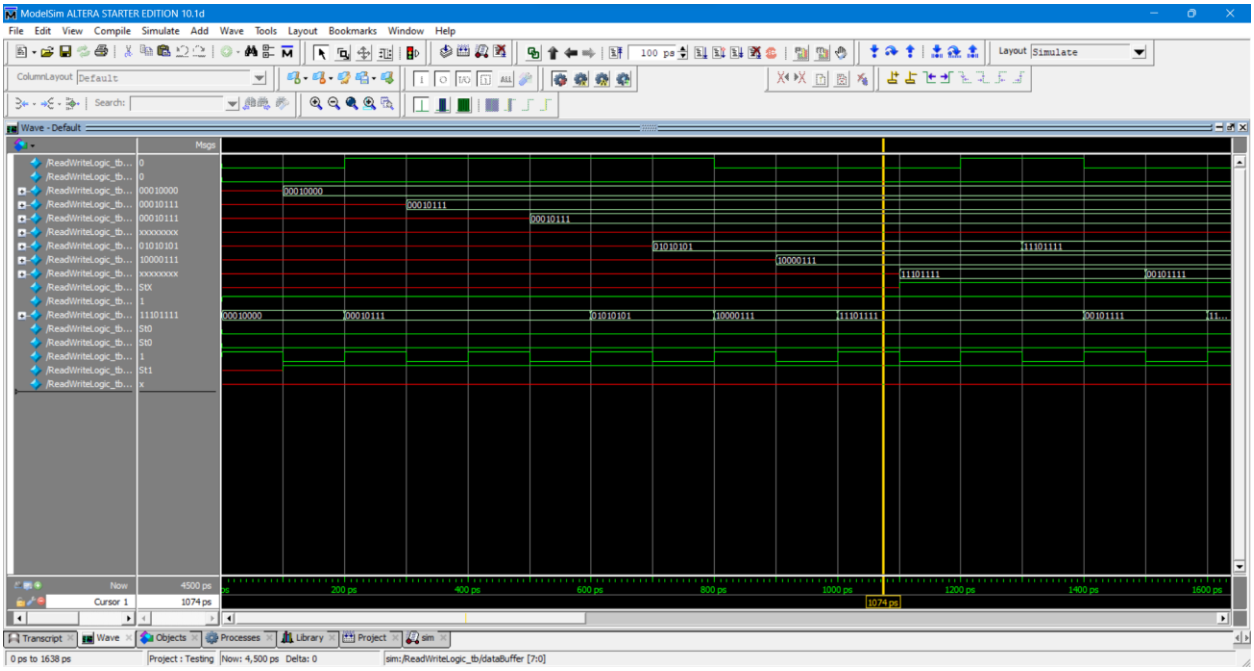

Figure 10: Read\Write Logic TB

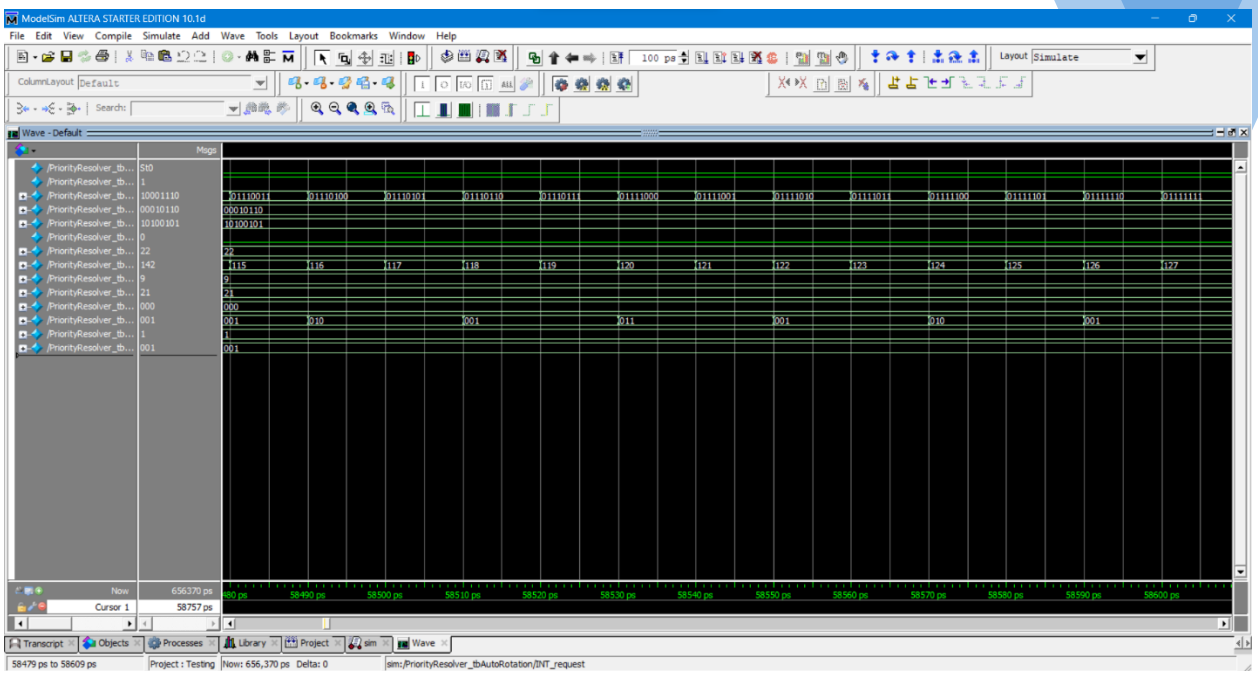

Figure 11: Priority Resolver Auto Rotation TB

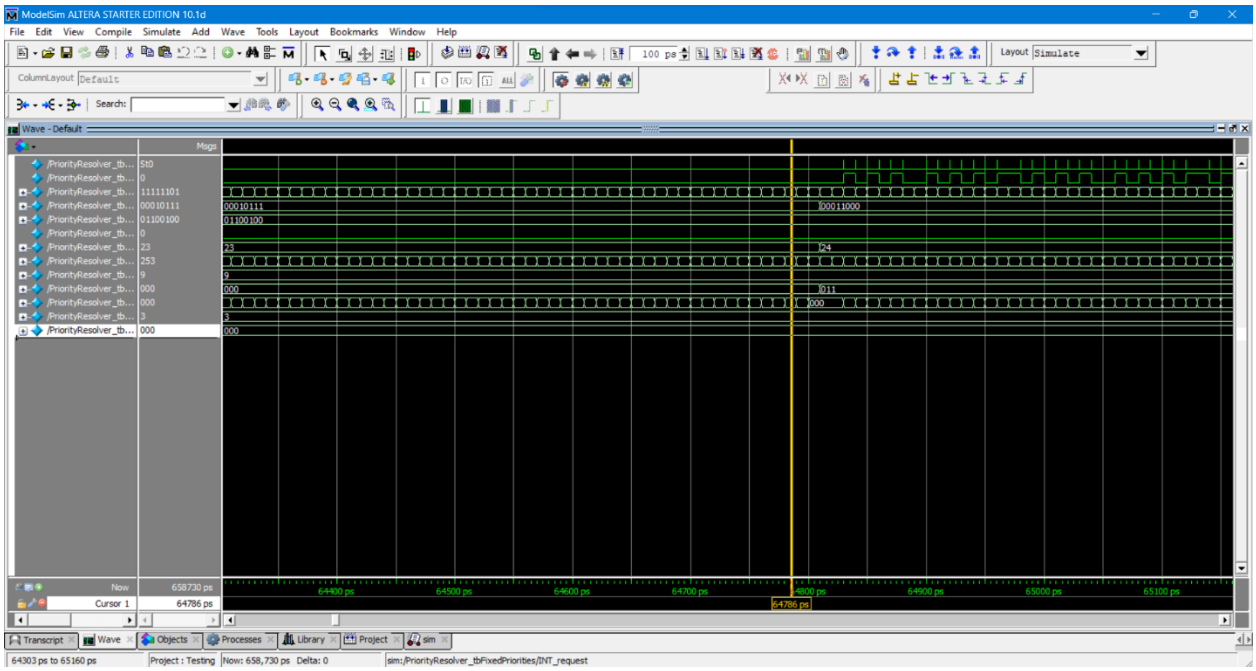

Figure 12: Priority Resolver Fixed Priorities TB

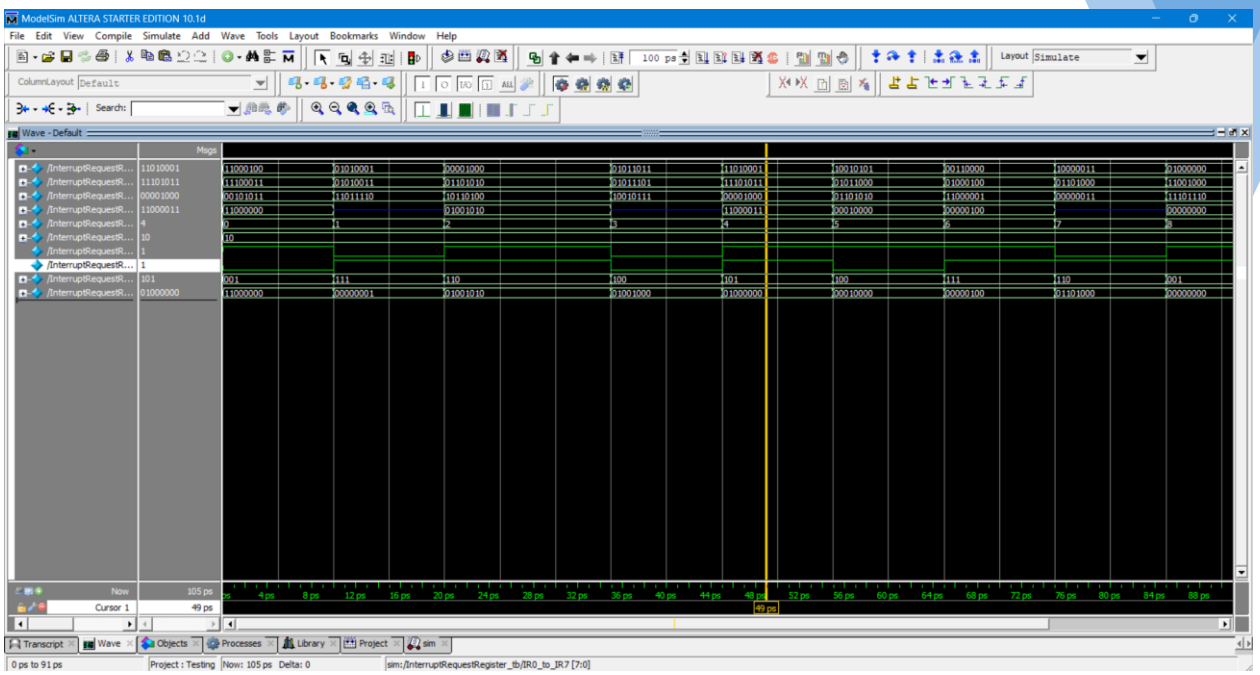

Figure 13: Interrupt Request Register TB

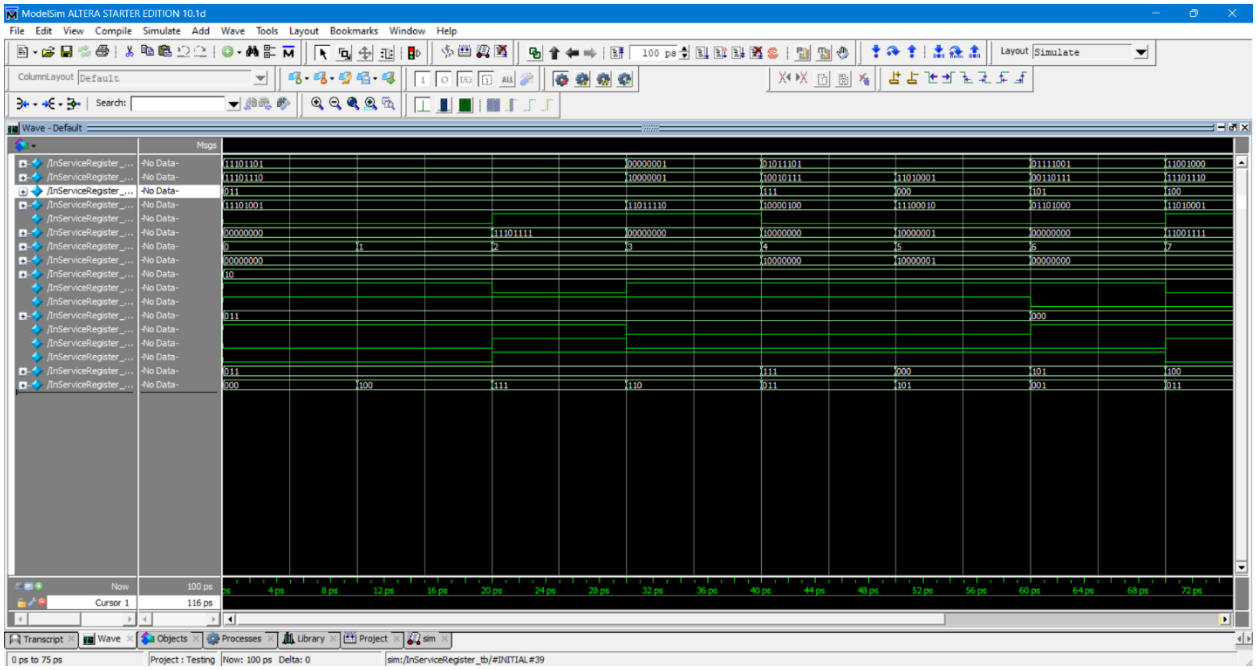

Figure 14: In Service Register TB

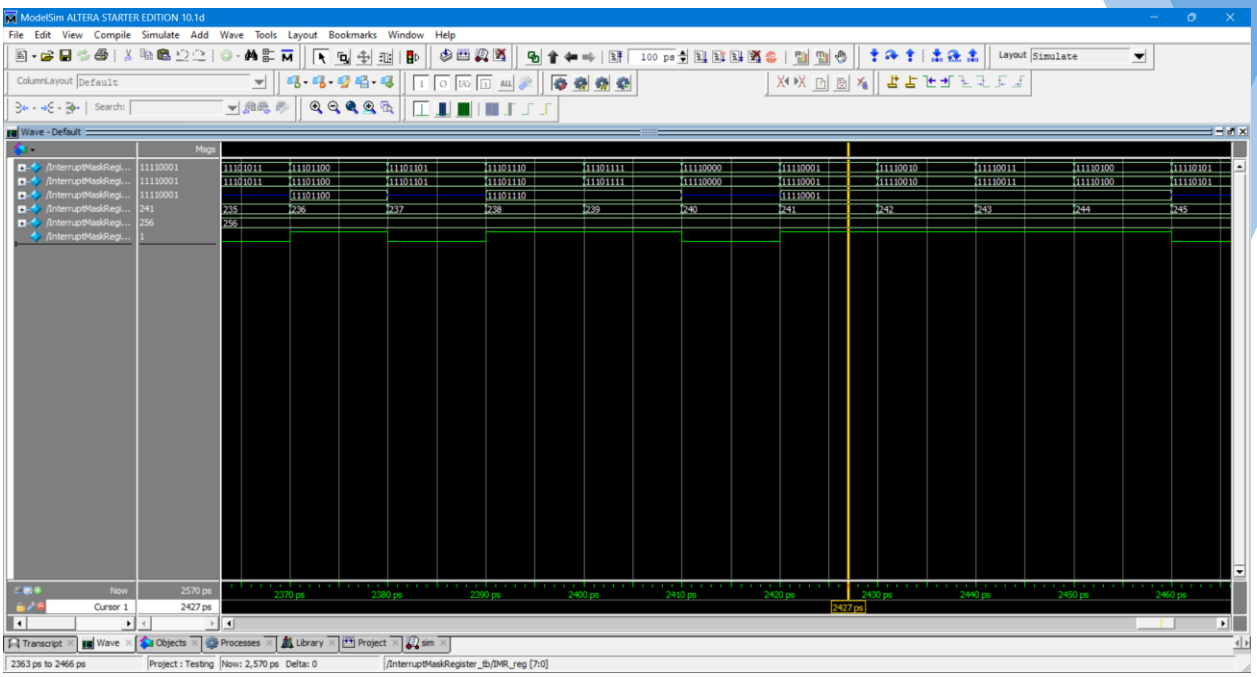

Figure 15: Interrupt Mask Register TB

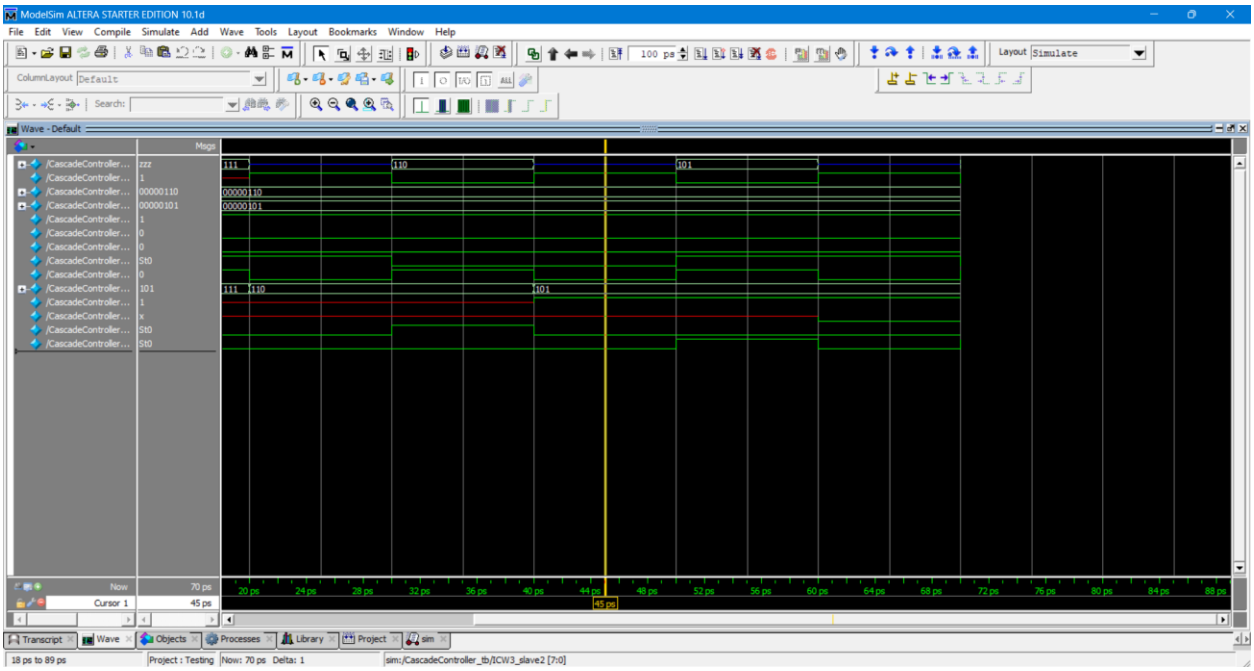

Figure 16: Cascade Controller TB

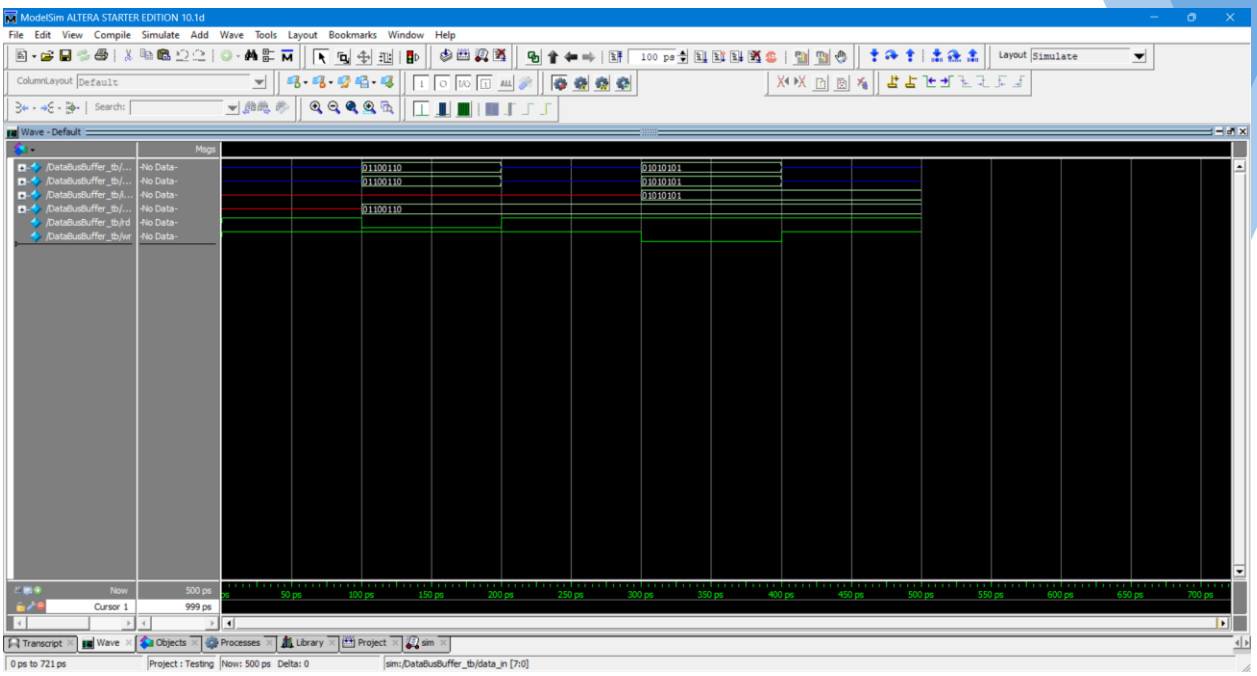

Figure 17: Data Bus Buffer TB

8.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	

9.References:

10. GitHub Repo:

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