8259A PIC Project

Team members

Name	ID	GitHub username	contribution
Abdullah mohamed	2001803	AntiHexCode	Control logic, Read Logic, PIC8259A, Testbench
Ahmad Mahfouz	2002238	rye141200	Control logic, Write logic PIC8259A, report
Mohamed Mostafa	2001299	mohamed-most	Interrupt logic PIC 8259A ,report

Table of contents:

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 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:

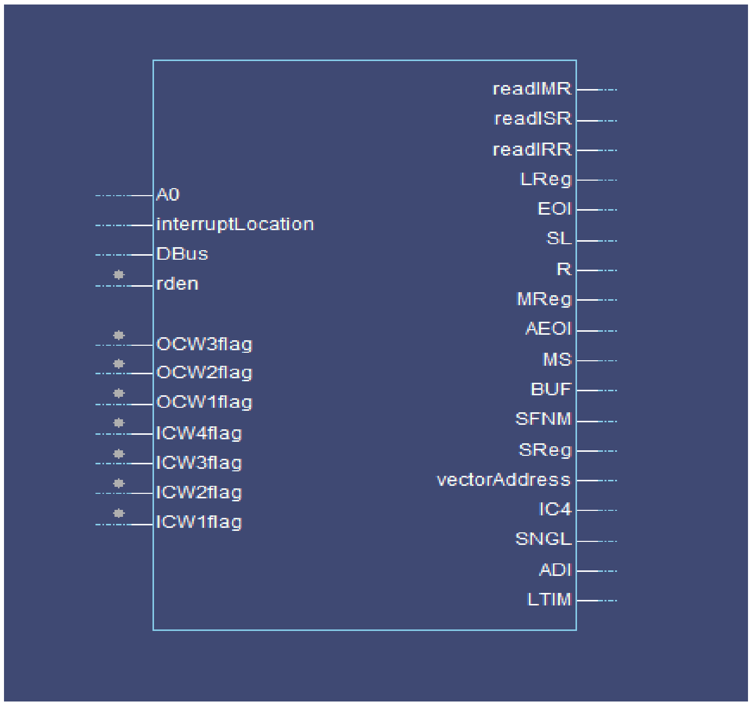
- 8086 compatible
- Programmable
- Single +5V supply, no master clock
- Eight-Level Priority Controller
- Expandable to 64 Levels via cascading
- Handling interrupts in fully-nested mode/automatic rotation
- Interrupt masking compatible
- EOI/AEOI supportive
- supportive for reading status

sequence of operation:

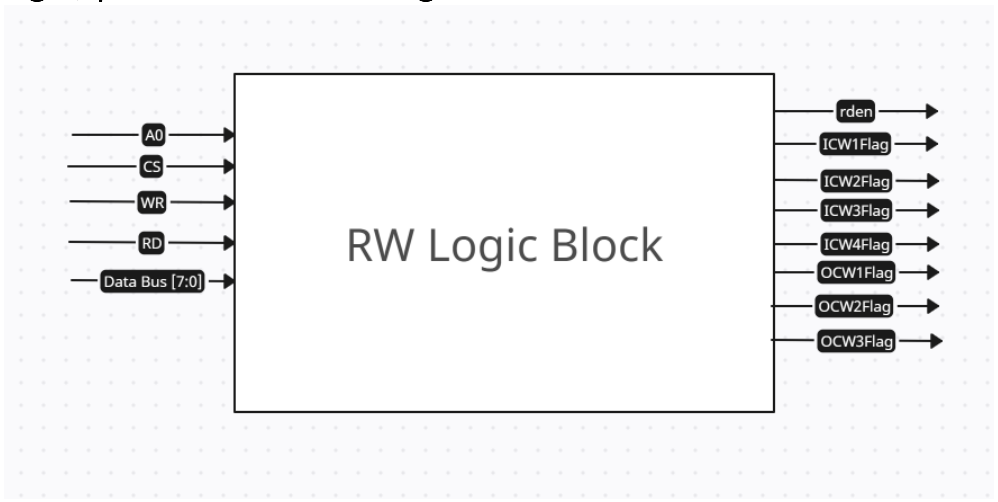
1. All command words are sent from 8086 to the RW logic.
2. RW logic parses the command words sending flags to control logic
3. Whilst command words are being sent, all blocks are initializing according to the command words
4. Once all command words are sent, other blocks can start working on the interrupt.
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

Block Diagrams:

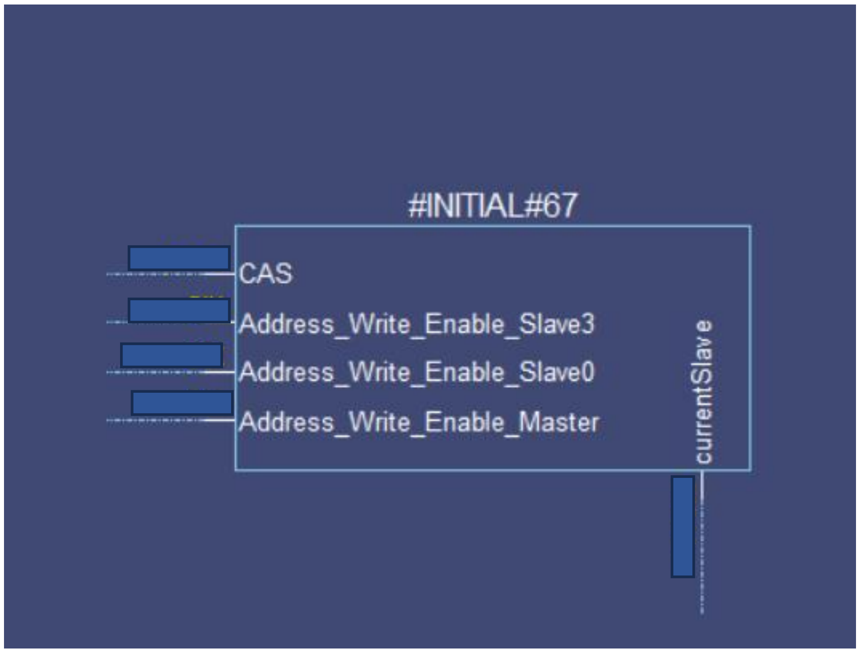
Control logic block diagram, the mastermind of the PIC, takes flags from R/W logic, parses the data to give it to other blocks.



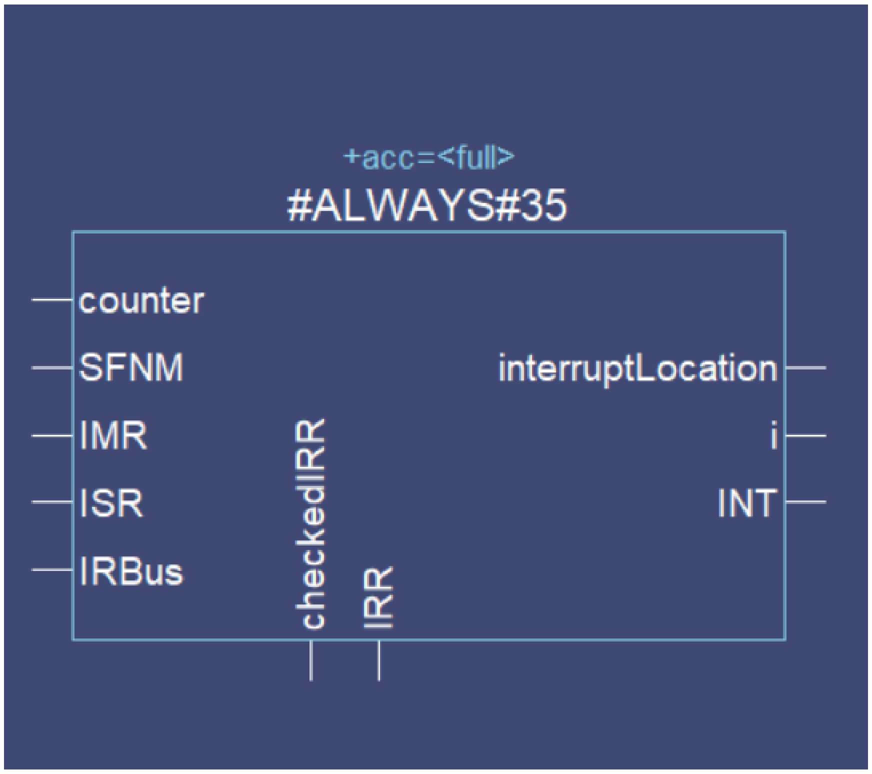
Control logic block diagram, the mastermind of the PIC, takes flags from R/W logic, parses the data to give it to other blocks



cascade logic block diagram:


#INITIAL#67
CAS
Address_Write_Enable_Slave3
Address_Write_Enable_Slave0
Address_Write_Enable_Master
currentSlave

Interrupt block diagram:



signals:

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

R/W logic signals

Signal	Description
A0	1 bit input from 8086, used to identify command words
CS	1 bit active low input from 8086, turns on the PIC or off
WR	1 bit active low input from 8086, when asserted, allows writing in RW logic
RD	1 bit active low input from 8086, when asserted, allows reading status of PIC
Data Bus	8 bit buffer, carries command words from 8086. Takes data from PIC to 8086. It is the main method of communication between 8086 and PIC
rden	1 bit output, used by control logic to let it know that read signal is asserted
ICW1Flag	1 bit output, a flag to indicate the current command word is ICW1
ICW2Flag	1 bit output, a flag to indicate the current command word is ICW2
ICW3Flag	1 bit output, a flag to indicate the current command word is ICW3
ICW4Flag	1 bit output, a flag to indicate the current command word is ICW4
OCW1Flag	1 bit output, a flag to indicate the current command word is OCW1
OCW2Flag	1 bit output, a flag to indicate the current command word is OCW2
OCW3Flag	1 bit output, a flag to indicate the current command word is OCW3

Cascade logic signals

Signal	Description
SP	input, 1'b1 if the module is a master, 1'b0 if the module is a slave and it's always an input for cascade mode (disregard the Buffer mode)
SNGL	input, 1'b1 if the module is in single mode, 1'b0 if the module is in cascade mode
ICW3	input, 8-bit vector that contains the interrupt address of the module
Interrupt_Location	input, 3-bit vector that contains the location of the current interrupting slave on the interrupt inputs for the IRR of the master
CAS (Input)	input, for slaves it's an input coming from the master and it's an acknowledgement from the master that the slave is enabled
CAS (Output)	output, for master it's an output that goes to the slaves and it's an acknowledgement from the master that the slave is enabled
Address_Write_Enable	output, 1'b1 if the module is allowed to send the interrupt address vector to the data bus, 1'b0 otherwise

Interrupt signals:

1. **IRBus (Interrupt Request Bus):**
   - This is an 8-bit input bus representing the interrupt request lines IR0 through IR7.
   - When any interrupt line is active (low), it indicates a request for service.
2. **LTIM (Level Triggered Interrupt Mode):**
   - Level Triggered Mode is a control signal that determines whether interrupts are level-sensitive or edge-sensitive.
   - If LTIM is high, it indicates level-triggered mode; otherwise, it's edge-triggered.
3. **SFNM (Special Fully Nested Mode):**
   - SFNM is an input that, when activated, enables special fully nested mode.
   - In this mode, the controller prioritizes pending interrupts more efficiently.
4. **AR (Automatic Rotation Mode):**
   - Automatic Rotation Mode, when enabled, automatically rotates priority among the interrupt lines.
5. **AEOI (Automatic End Of Interrupt):**
   - AEOI, when set, allows the 8259 PIC to automatically issue an End of Interrupt (EOI) signal after servicing an interrupt.
6. **TReg (Task Priority Register):**
   - TReg is a 5-bit input used for specifying the interrupt vector address during interrupt acknowledgment.
7. **IMR (Interrupt Mask Register):**
   - IMR is an 8-bit input where each bit corresponds to an interrupt line. A high bit masks (disables) the corresponding interrupt line.
8. **readIRR (Read Interrupt Request Register):**
   - When readIRR is high, it indicates a request to read the Interrupt Request Register (IRR) to check which interrupts are pending.
9. **readISR (Read Interrupt Service Register):**
   - When readISR is high, it indicates a request to read the Interrupt Service Register (ISR) to check which interrupts are being serviced.
10. **INTA (Interrupt Acknowledge):**
    - INTA is an input used to acknowledge the receipt of an interrupt. It is active low.
11. **INT (Interrupt):**
    - INT is an output signal indicating that an interrupt is being serviced.
12. **internalBus:**

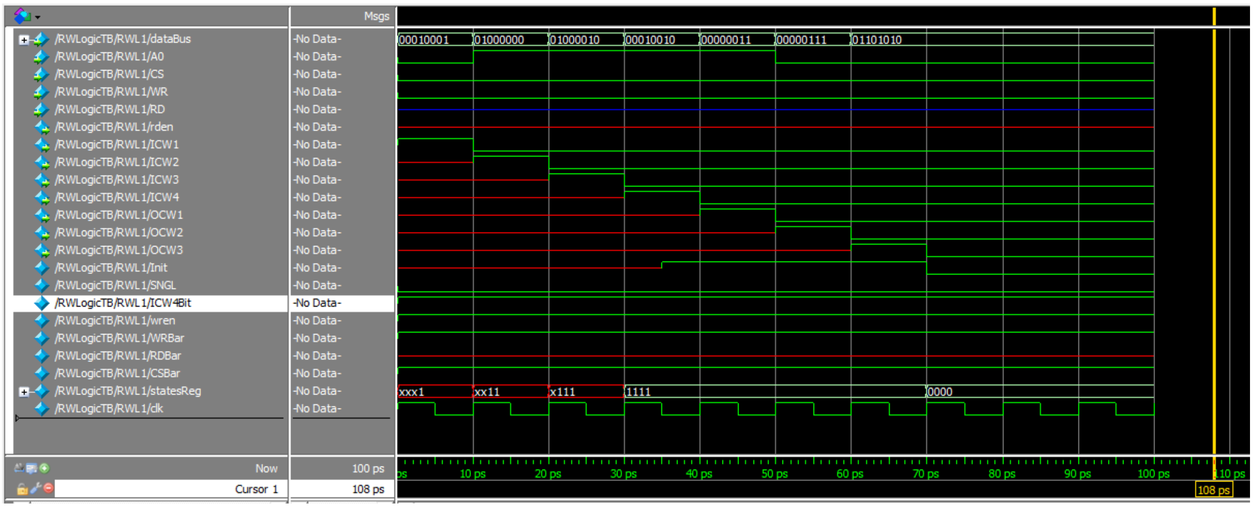
- An 8-bit output representing the internal bus used for data transfer within the PIC.

13. **interruptLocation:**
    - A 3-bit output representing the location of the active interrupt within the PIC.
14. **IRR (Interrupt Request Register):**
    - An 8-bit output representing the state of interrupt requests, indicating which interrupts are pending.

simulation:

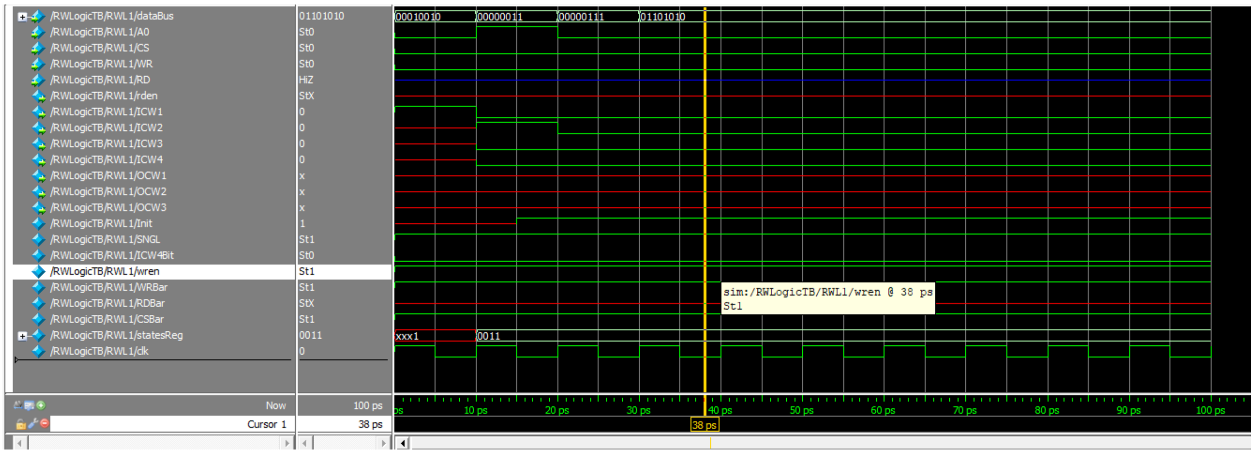
R/W logic simulation:

- All commands are written



```
                              Msgs
/RWLogicTB/RWL1/dataBus       -No Data-
/RWLogicTB/RWL1/A0            -No Data-
/RWLogicTB/RWL1/CS            -No Data-
/RWLogicTB/RWL1/WR            -No Data-
/RWLogicTB/RWL1/RD            -No Data-
/RWLogicTB/RWL1/rden          -No Data-
/RWLogicTB/RWL1/ICW1          -No Data-
/RWLogicTB/RWL1/ICW2          -No Data-
/RWLogicTB/RWL1/ICW3          -No Data-
/RWLogicTB/RWL1/ICW4          -No Data-
/RWLogicTB/RWL1/OCW1          -No Data-
/RWLogicTB/RWL1/OCW2          -No Data-
/RWLogicTB/RWL1/OCW3          -No Data-
/RWLogicTB/RWL1/Init          -No Data-
/RWLogicTB/RWL1/SNGL          -No Data-
/RWLogicTB/RWL1/ICW4Bit       -No Data-
/RWLogicTB/RWL1/wren          -No Data-
/RWLogicTB/RWL1/WRBar         -No Data-
/RWLogicTB/RWL1/RDBar         -No Data-
/RWLogicTB/RWL1/CSBar         -No Data-
/RWLogicTB/RWL1/statesReg     -No Data-
/RWLogicTB/RWL1/clk           -No Data-
Now                           100 ps
Cursor 1                      108 ps
```

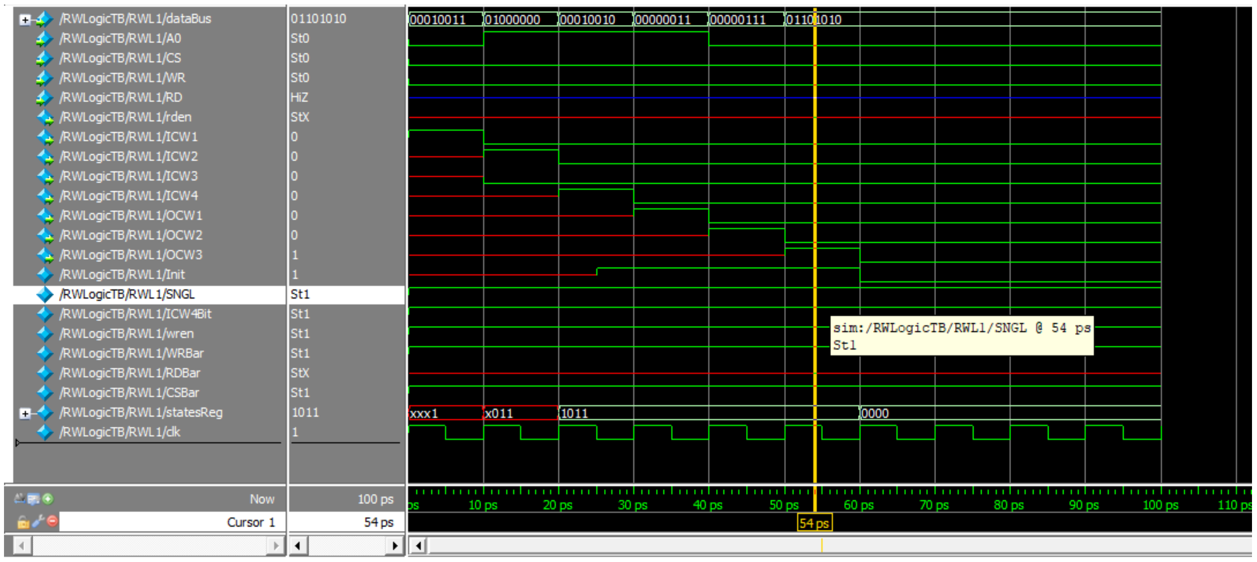
- ICW3 and ICW4 aren't written.



```
/RWLogicTB/RWL1/dataBus       01101010
/RWLogicTB/RWL1/A0            St0
/RWLogicTB/RWL1/CS            St0
/RWLogicTB/RWL1/WR            St0
/RWLogicTB/RWL1/RD            HiZ
/RWLogicTB/RWL1/rden          StX
/RWLogicTB/RWL1/ICW1          0
/RWLogicTB/RWL1/ICW2          0
/RWLogicTB/RWL1/ICW3          0
/RWLogicTB/RWL1/ICW4          0
/RWLogicTB/RWL1/OCW1          x
/RWLogicTB/RWL1/OCW2          x
/RWLogicTB/RWL1/OCW3          x
/RWLogicTB/RWL1/Init          1
/RWLogicTB/RWL1/SNGL          St1
/RWLogicTB/RWL1/ICW4Bit       St0
/RWLogicTB/RWL1/wren          St1
/RWLogicTB/RWL1/WRBar         St1
/RWLogicTB/RWL1/RDBar         StX
/RWLogicTB/RWL1/CSBar         St1
/RWLogicTB/RWL1/statesReg     0011
/RWLogicTB/RWL1/clk           0
Now                           100 ps
Cursor 1                      38 ps
```

```
sim:/RWLogicTB/RWL1/wren @ 38 ps
St1
```

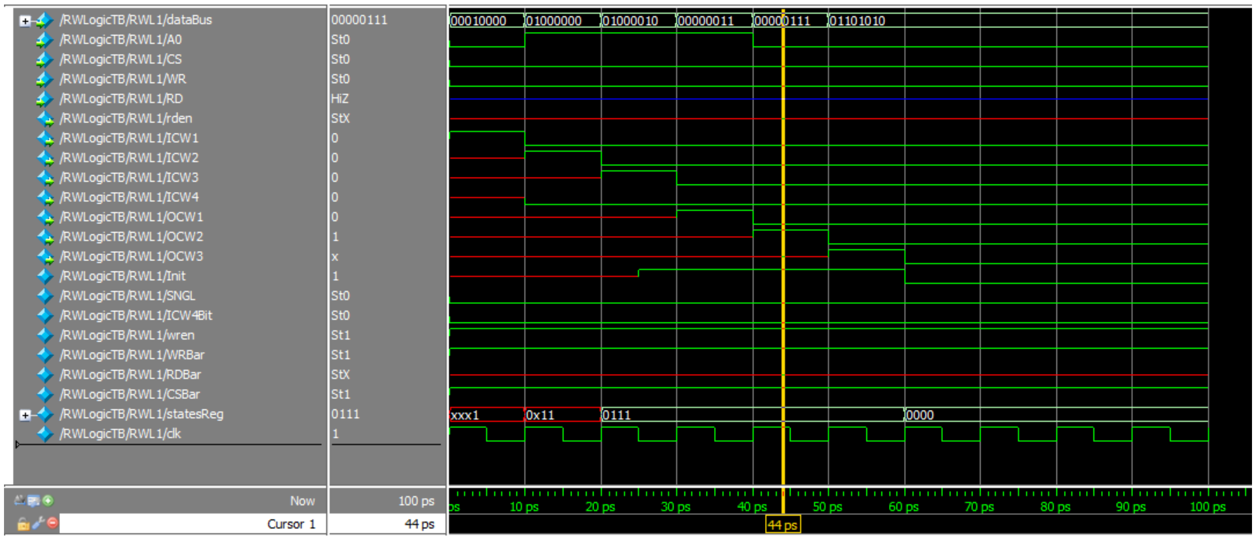
- ICW3 isn't written.



/RWLogicTB/RWL1/dataBus	01101010
/RWLogicTB/RWL1/A0	St0
/RWLogicTB/RWL1/CS	St0
/RWLogicTB/RWL1/WR	St0
/RWLogicTB/RWL1/RD	HiZ
/RWLogicTB/RWL1/rden	StX
/RWLogicTB/RWL1/ICW1	0
/RWLogicTB/RWL1/ICW2	0
/RWLogicTB/RWL1/ICW3	0
/RWLogicTB/RWL1/ICW4	0
/RWLogicTB/RWL1/OCW1	0
/RWLogicTB/RWL1/OCW2	0
/RWLogicTB/RWL1/OCW3	1
/RWLogicTB/RWL1/Init	1
/RWLogicTB/RWL1/SNGL	St1
/RWLogicTB/RWL1/ICW4Bit	St1
/RWLogicTB/RWL1/wren	St1
/RWLogicTB/RWL1/WRBar	St1
/RWLogicTB/RWL1/RDBar	StX
/RWLogicTB/RWL1/CSBar	St1
/RWLogicTB/RWL1/statesReg	1011
/RWLogicTB/RWL1/clk	1
Now	100 ps
Cursor 1	54 ps

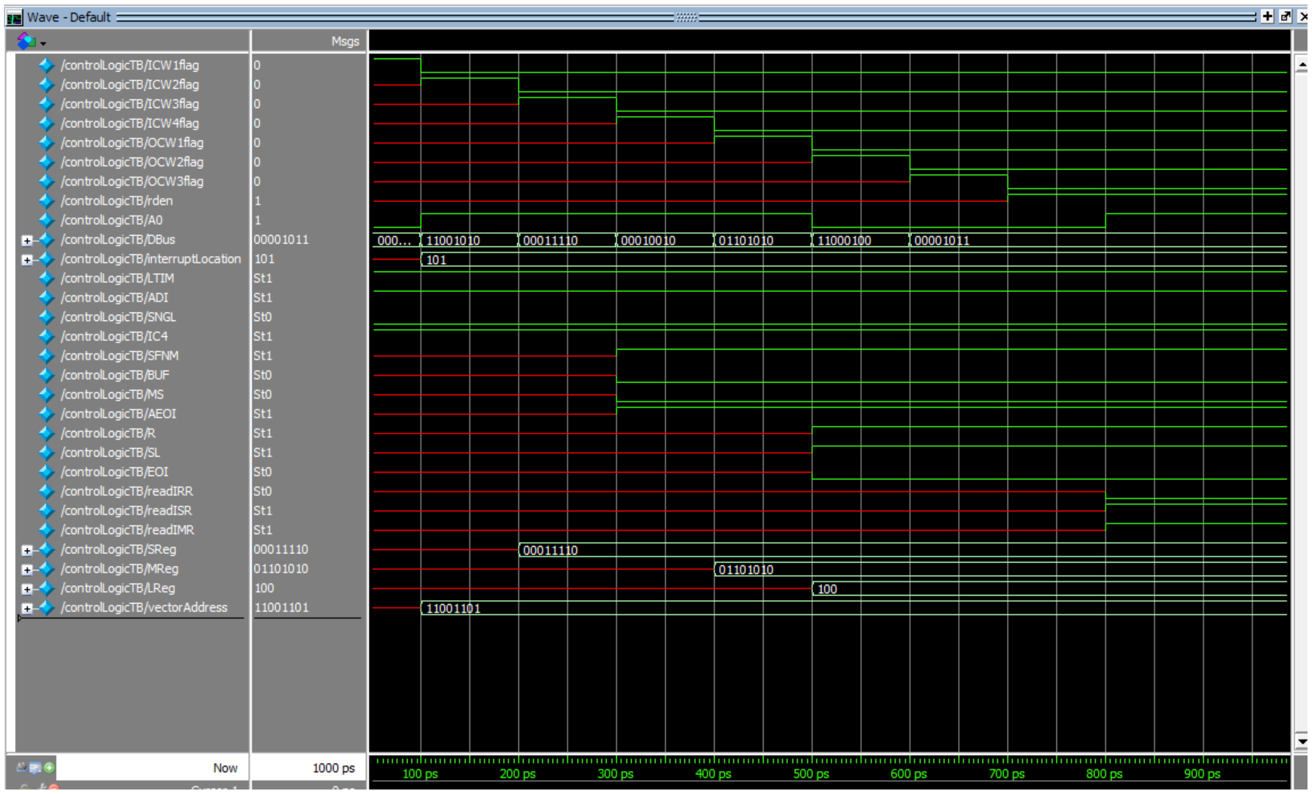
```
sim:/RWLogicTB/RWL1/SNGL @ 54 ps
St1
```

- ICW4 isn't written.

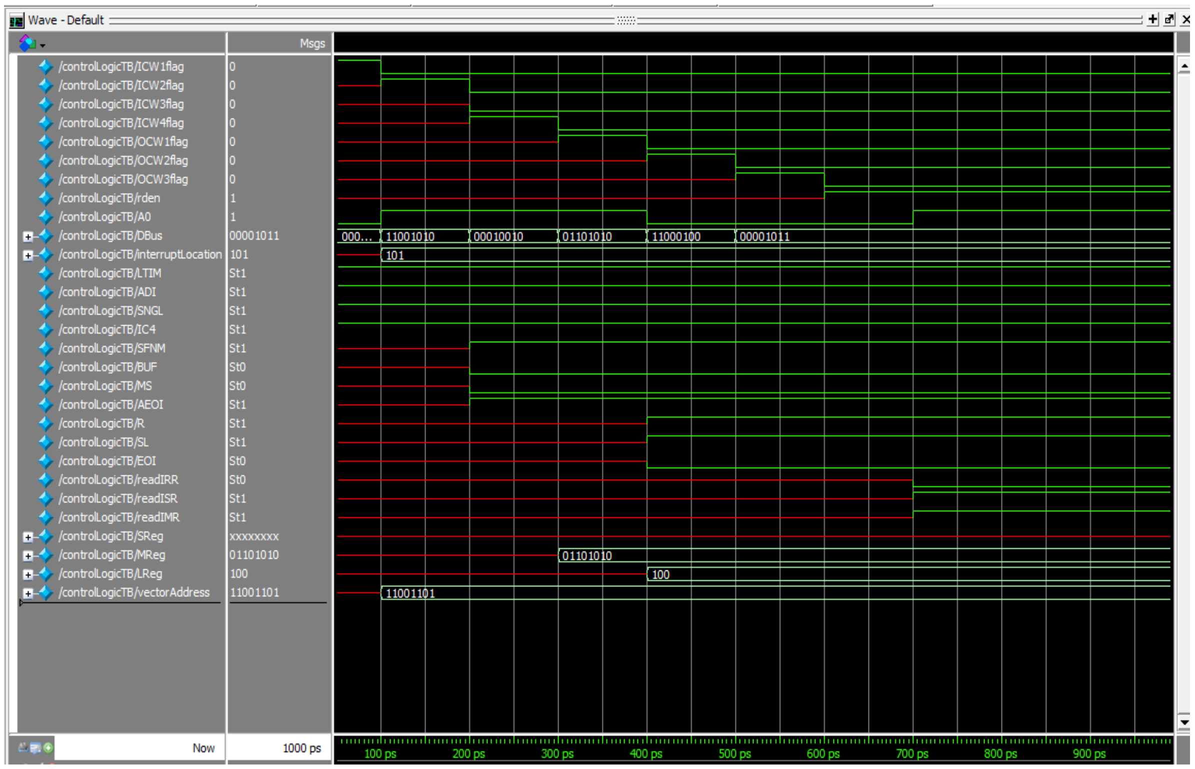


/RWLogicTB/RWL1/dataBus	00000111
/RWLogicTB/RWL1/A0	St0
/RWLogicTB/RWL1/CS	St0
/RWLogicTB/RWL1/WR	St0
/RWLogicTB/RWL1/RD	HiZ
/RWLogicTB/RWL1/rden	StX
/RWLogicTB/RWL1/ICW1	0
/RWLogicTB/RWL1/ICW2	0
/RWLogicTB/RWL1/ICW3	0
/RWLogicTB/RWL1/ICW4	0
/RWLogicTB/RWL1/OCW1	0
/RWLogicTB/RWL1/OCW2	1
/RWLogicTB/RWL1/OCW3	x
/RWLogicTB/RWL1/Init	1
/RWLogicTB/RWL1/SNGL	St0
/RWLogicTB/RWL1/ICW4Bit	St0
/RWLogicTB/RWL1/wren	St1
/RWLogicTB/RWL1/WRBar	St1
/RWLogicTB/RWL1/RDBar	StX
/RWLogicTB/RWL1/CSBar	St1
/RWLogicTB/RWL1/statesReg	0111
/RWLogicTB/RWL1/clk	1
Now	100 ps
Cursor 1	44 ps

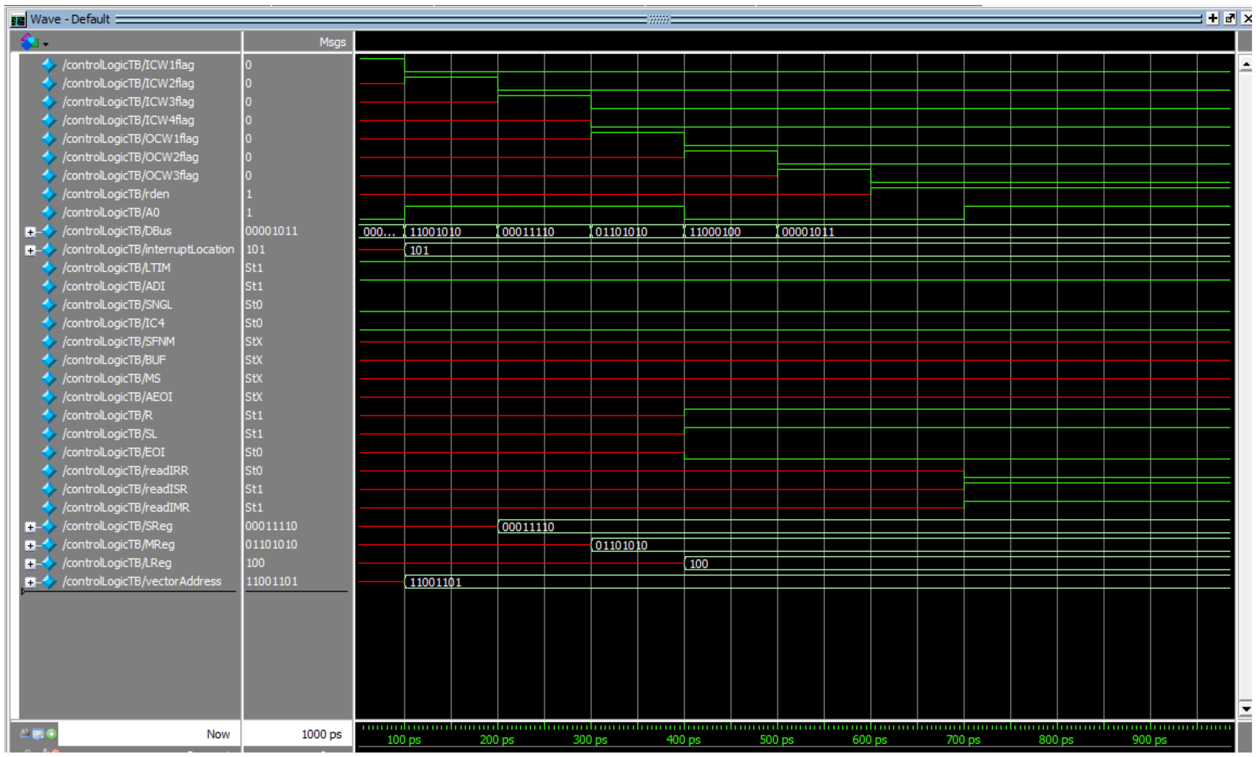
Control logic simulation:



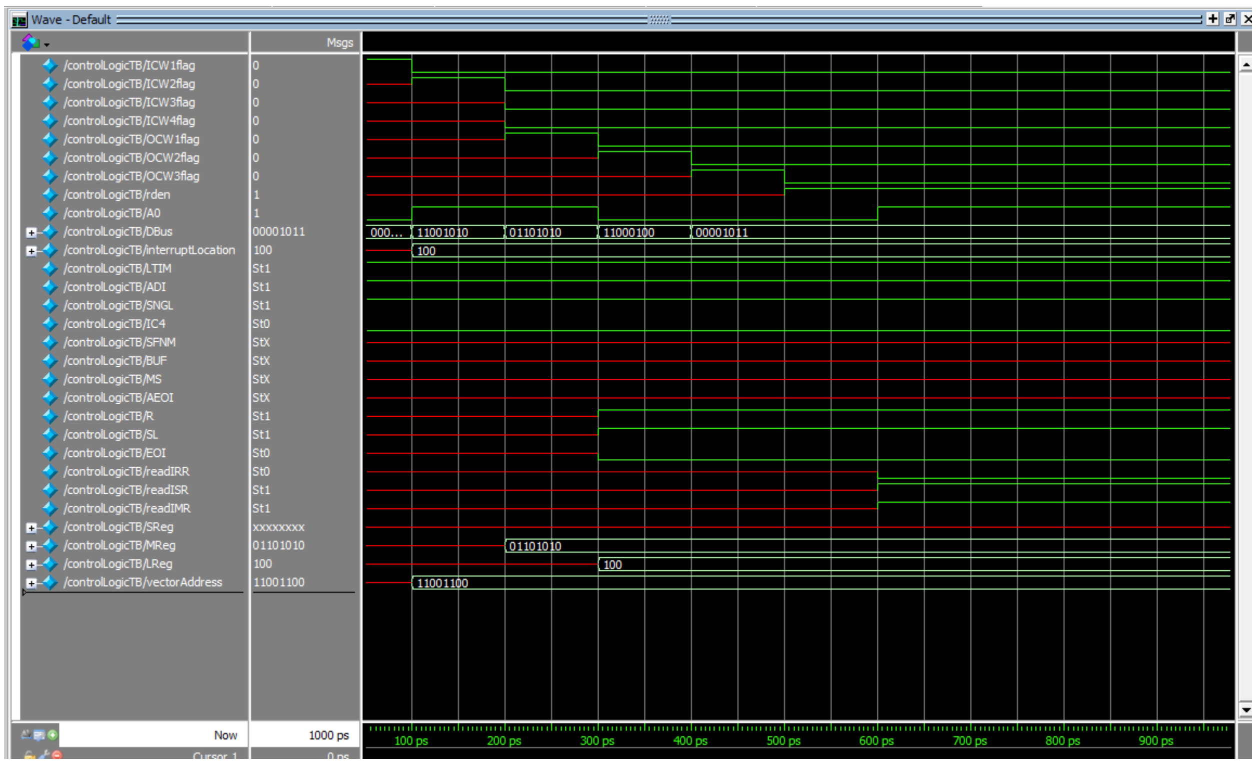
	Msgs
/controlLogicTB/ICW1flag	0
/controlLogicTB/ICW2flag	0
/controlLogicTB/ICW3flag	0
/controlLogicTB/ICW4flag	0
/controlLogicTB/OCW1flag	0
/controlLogicTB/OCW2flag	0
/controlLogicTB/OCW3flag	0
/controlLogicTB/rden	1
/controlLogicTB/A0	1
/controlLogicTB/DBus	00001011
/controlLogicTB/interruptLocation	101
/controlLogicTB/LTIM	St1
/controlLogicTB/ADI	St1
/controlLogicTB/SNGL	St0
/controlLogicTB/IC4	St1
/controlLogicTB/SFNM	St1
/controlLogicTB/BUF	St0
/controlLogicTB/MS	St0
/controlLogicTB/AEOI	St1
/controlLogicTB/R	St1
/controlLogicTB/SL	St1
/controlLogicTB/EOI	St0
/controlLogicTB/readIRR	St0
/controlLogicTB/readISR	St1
/controlLogicTB/readIMR	St1
/controlLogicTB/SReg	00011110
/controlLogicTB/MReg	01101010
/controlLogicTB/LReg	100
/controlLogicTB/vectorAddress	11001101



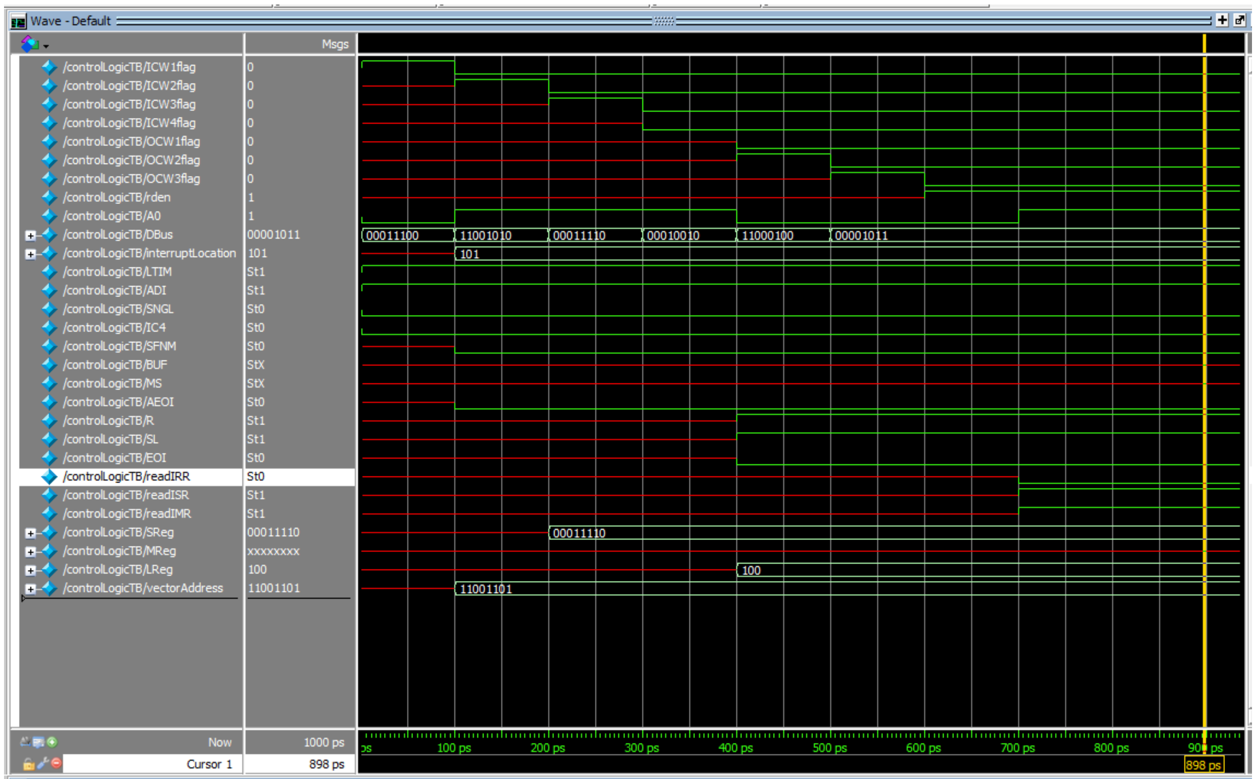
	Msgs
/controlLogicTB/ICW1flag	0
/controlLogicTB/ICW2flag	0
/controlLogicTB/ICW3flag	0
/controlLogicTB/ICW4flag	0
/controlLogicTB/OCW1flag	0
/controlLogicTB/OCW2flag	0
/controlLogicTB/OCW3flag	0
/controlLogicTB/rden	1
/controlLogicTB/A0	1
/controlLogicTB/DBus	00001011
/controlLogicTB/interruptLocation	101
/controlLogicTB/LTIM	St1
/controlLogicTB/ADI	St1
/controlLogicTB/SNGL	St1
/controlLogicTB/IC4	St1
/controlLogicTB/SFNM	St1
/controlLogicTB/BUF	St0
/controlLogicTB/MS	St0
/controlLogicTB/AEOI	St1
/controlLogicTB/R	St1
/controlLogicTB/SL	St1
/controlLogicTB/EOI	St0
/controlLogicTB/readIRR	St0
/controlLogicTB/readISR	St1
/controlLogicTB/readIMR	St1
/controlLogicTB/SReg	xxxxxxxx
/controlLogicTB/MReg	01101010
/controlLogicTB/LReg	100
/controlLogicTB/vectorAddress	11001101



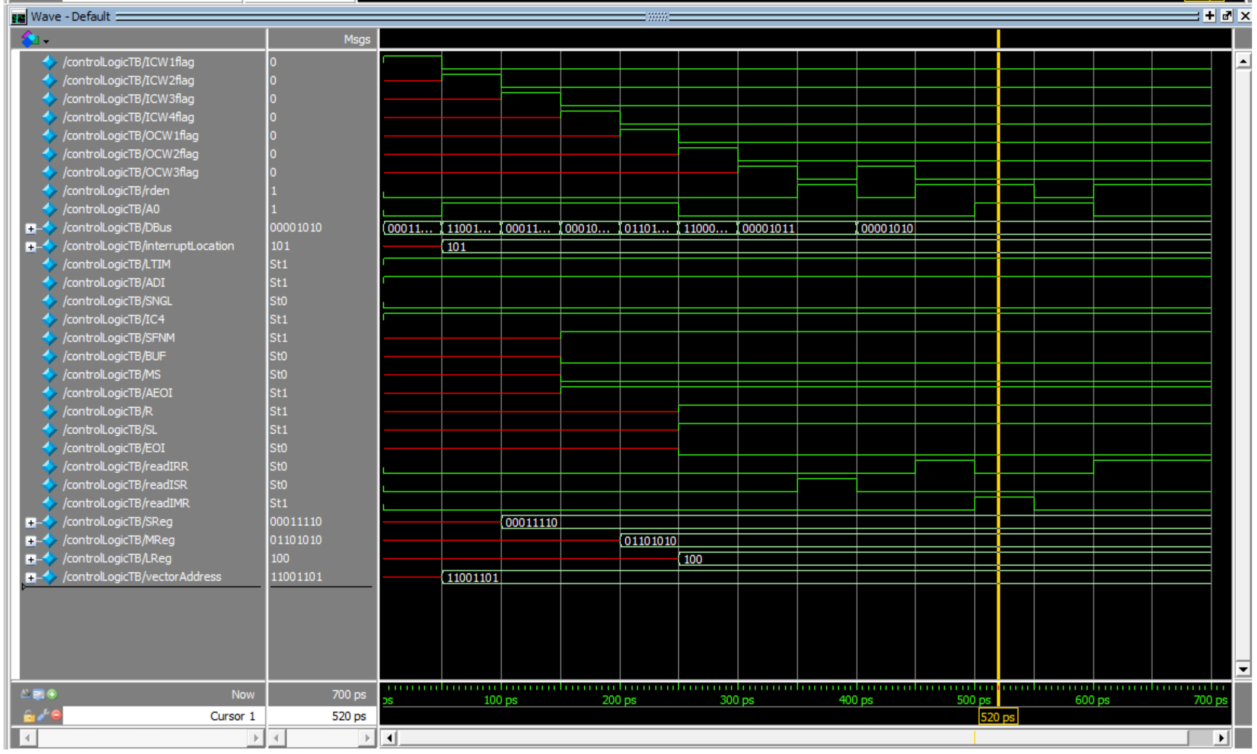
```
/controlLogicTB/ICW1flag          0
/controlLogicTB/ICW2flag          0
/controlLogicTB/ICW3flag          0
/controlLogicTB/ICW4flag          0
/controlLogicTB/OCW1flag          0
/controlLogicTB/OCW2flag          0
/controlLogicTB/OCW3flag          0
/controlLogicTB/rden              1
/controlLogicTB/A0                1
/controlLogicTB/DBus              00001011
/controlLogicTB/interruptLocation 101
/controlLogicTB/LTIM              St1
/controlLogicTB/ADI               St1
/controlLogicTB/SNGL              St0
/controlLogicTB/IC4               St0
/controlLogicTB/SFNM              StX
/controlLogicTB/BUF               StX
/controlLogicTB/MS                StX
/controlLogicTB/AEOI              StX
/controlLogicTB/R                 St1
/controlLogicTB/SL                St1
/controlLogicTB/EOI               St0
/controlLogicTB/readIRR           St0
/controlLogicTB/readISR           St1
/controlLogicTB/readIMR           St1
/controlLogicTB/SReg              00011110
/controlLogicTB/MReg              01101010
/controlLogicTB/LReg              100
/controlLogicTB/vectorAddress     11001101
Now                               1000 ps
```



```
/controlLogicTB/ICW1flag          0
/controlLogicTB/ICW2flag          0
/controlLogicTB/ICW3flag          0
/controlLogicTB/ICW4flag          0
/controlLogicTB/OCW1flag          0
/controlLogicTB/OCW2flag          0
/controlLogicTB/OCW3flag          0
/controlLogicTB/rden              1
/controlLogicTB/A0                1
/controlLogicTB/DBus              00001011
/controlLogicTB/interruptLocation 100
/controlLogicTB/LTIM              St1
/controlLogicTB/ADI               St1
/controlLogicTB/SNGL              St1
/controlLogicTB/IC4               St0
/controlLogicTB/SFNM              StX
/controlLogicTB/BUF               StX
/controlLogicTB/MS                StX
/controlLogicTB/AEOI              StX
/controlLogicTB/R                 St1
/controlLogicTB/SL                St1
/controlLogicTB/EOI               St0
/controlLogicTB/readIRR           St0
/controlLogicTB/readISR           St1
/controlLogicTB/readIMR           St1
/controlLogicTB/SReg              xxxxxxxx
/controlLogicTB/MReg              01101010
/controlLogicTB/LReg              100
/controlLogicTB/vectorAddress     11001100
Now                               1000 ps
Cursor 1                          0 ps
```

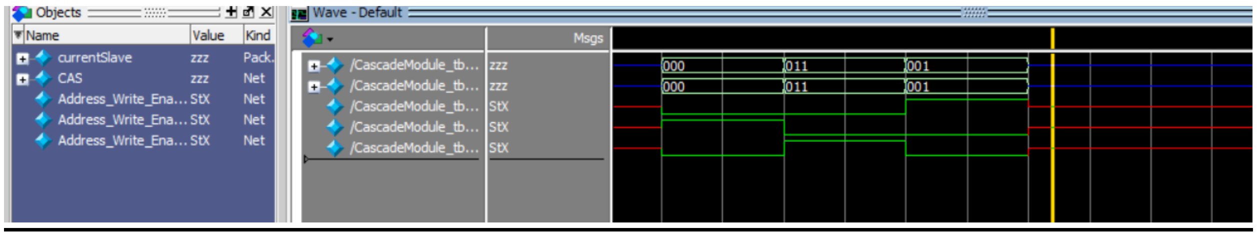


```
Wave - Default
                                   Msgs
/controlLogicTB/ICW1flag           0
/controlLogicTB/ICW2flag           0
/controlLogicTB/ICW3flag           0
/controlLogicTB/ICW4flag           0
/controlLogicTB/OCW1flag           0
/controlLogicTB/OCW2flag           0
/controlLogicTB/OCW3flag           0
/controlLogicTB/rden               1
/controlLogicTB/A0                 1
/controlLogicTB/DBus               00001011
/controlLogicTB/interruptLocation  101
/controlLogicTB/LTIM               St1
/controlLogicTB/ADI                St1
/controlLogicTB/SNGL               St0
/controlLogicTB/IC4                St0
/controlLogicTB/SFNM               St0
/controlLogicTB/BUF                StX
/controlLogicTB/MS                 StX
/controlLogicTB/AEOI               St0
/controlLogicTB/R                  St1
/controlLogicTB/SL                 St1
/controlLogicTB/EOI                St0
/controlLogicTB/readIRR            St0
/controlLogicTB/readISR            St1
/controlLogicTB/readIMR            St1
/controlLogicTB/SReg               00011110
/controlLogicTB/MReg               xxxxxxxx
/controlLogicTB/LReg               100
/controlLogicTB/vectorAddress      11001101
Now                                1000 ps
Cursor 1                           898 ps
```



```
Wave - Default
                                   Msgs
/controlLogicTB/ICW1flag           0
/controlLogicTB/ICW2flag           0
/controlLogicTB/ICW3flag           0
/controlLogicTB/ICW4flag           0
/controlLogicTB/OCW1flag           0
/controlLogicTB/OCW2flag           0
/controlLogicTB/OCW3flag           0
/controlLogicTB/rden               1
/controlLogicTB/A0                 1
/controlLogicTB/DBus               00001010
/controlLogicTB/interruptLocation  101
/controlLogicTB/LTIM               St1
/controlLogicTB/ADI                St1
/controlLogicTB/SNGL               St0
/controlLogicTB/IC4                St1
/controlLogicTB/SFNM               St1
/controlLogicTB/BUF                St0
/controlLogicTB/MS                 St0
/controlLogicTB/AEOI               St1
/controlLogicTB/R                  St1
/controlLogicTB/SL                 St1
/controlLogicTB/EOI                St0
/controlLogicTB/readIRR            St0
/controlLogicTB/readISR            St0
/controlLogicTB/readIMR            St1
/controlLogicTB/SReg               00011110
/controlLogicTB/MReg               01101010
/controlLogicTB/LReg               100
/controlLogicTB/vectorAddress      11001101
Now                                700 ps
Cursor 1                           520 ps
```

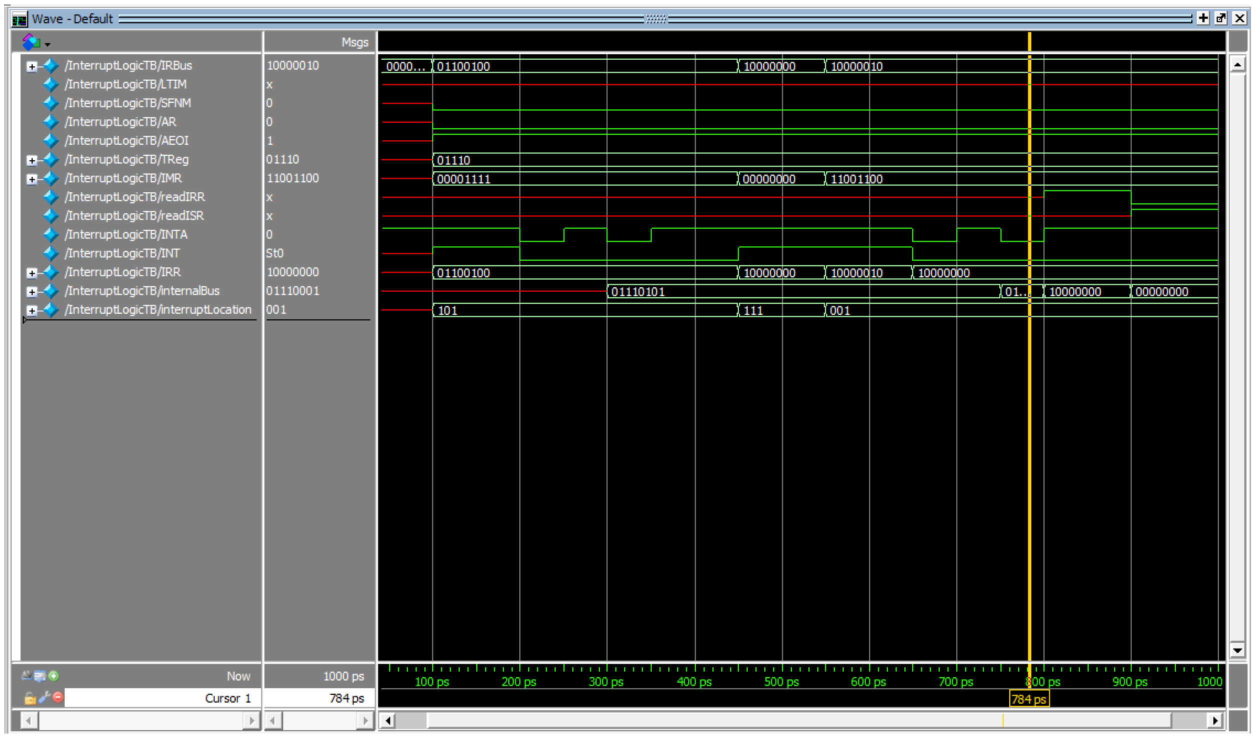
Cascade simulation:



Name	Value	Kind
currentSlave	zzz	Pack.
CAS	zzz	Net
Address_Write_Ena…	StX	Net
Address_Write_Ena…	StX	Net
Address_Write_Ena…	StX	Net

	Msgs
/CascadeModule_tb…	zzz
/CascadeModule_tb…	zzz
/CascadeModule_tb…	StX
/CascadeModule_tb…	StX
/CascadeModule_tb…	StX

Interrupt block simulation:



	Msgs
/InterruptLogicTB/IRBus	10000010
/InterruptLogicTB/LTIM	x
/InterruptLogicTB/SFNM	0
/InterruptLogicTB/AR	0
/InterruptLogicTB/AEOI	1
/InterruptLogicTB/TReg	01110
/InterruptLogicTB/IMR	11001100
/InterruptLogicTB/readIRR	x
/InterruptLogicTB/readISR	x
/InterruptLogicTB/INTA	0
/InterruptLogicTB/INT	St0
/InterruptLogicTB/IRR	10000000
/InterruptLogicTB/internalBus	01110001
/InterruptLogicTB/interruptLocation	001

Now	1000 ps
Cursor 1	784 ps

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.