SVA Specification

Contents

[Design 2](#_Toc161669904)

[SVA 2](#_Toc161669905)

[Testbench 5](#_Toc161669906)

[Interface 5](#_Toc161669907)

[Input interface 5](#_Toc161669908)

[Output interface 5](#_Toc161669909)

[Test 6](#_Toc161669910)

[Item 6](#_Toc161669911)

[Input item 6](#_Toc161669912)

[Output item 6](#_Toc161669913)

[Sequence 7](#_Toc161669914)

[Input sequence 7](#_Toc161669915)

[Output sequence 7](#_Toc161669916)

[Virtual sequence 7](#_Toc161669917)

[Environment 7](#_Toc161669918)

[Virtual sequencer 7](#_Toc161669919)

[Active Agent 7](#_Toc161669920)

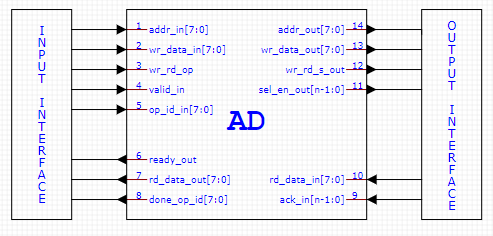
[Reactive Agent 7](#_Toc161669921)

[Scoreboard 8](#_Toc161669922)

[Coverage 8](#_Toc161669923)

[Display 8](#_Toc161669924)

# Design



Pe schema lipseste enable\_in pe interfata de input.

sel\_en\_out[4:0] nu ar trebui sa aiba aceeasi marime ca ack\_in[7:0]?

addr\_out e necesar, considerand ca avem sel\_en\_output? Sau invers?

# SVA

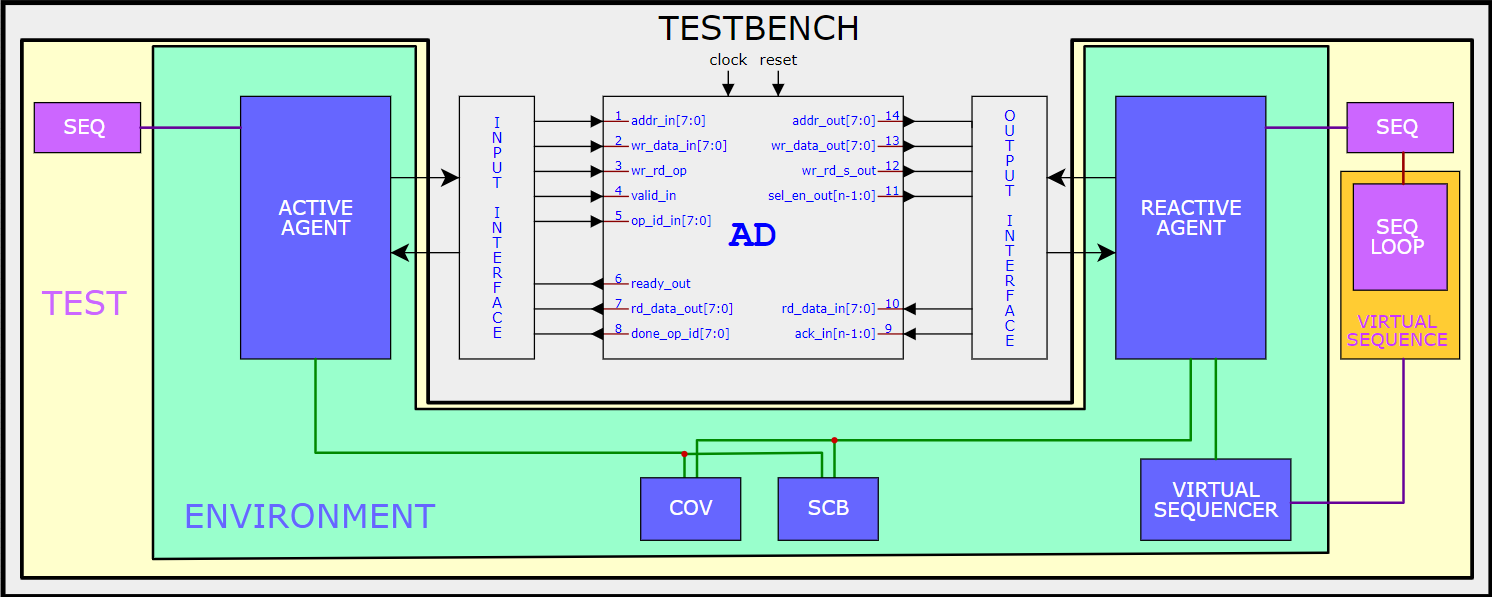
1. Reset case.
2. Read case on input.
3. Read case on output.
4. Write case on input.
5. Write case on output.
6. FIFO full and receives write.
7. FIFO full and receives output write.
8. FIFO almost full and receives write.
9. FIFO empty and receives output write.
10. Address out of bounds.
11. If reset then in the next cycle all output signals have to be 0/LOW.
12. If enable\_in HIGH, in the next cycle wr\_rd\_op LOW, valid\_in HIGH, addr\_in[7:0] and op\_id\_in[7:0] valid then done\_op\_id[7:0] and rd\_data\_out[7:0] are received in the same future cycle and equal the input.
13. If wr\_rd\_s\_out LOW, add\_out[7:0] and sel\_en\_out[4:0] valid then in the next cycle rd\_data\_in[7:0] valid and ack\_in[7:0] is 1 bit HIGH.
14. If enable\_in HIGH, next cycle valid\_in HIGH, wr\_rd\_op HIGH, addr\_in[7:0], rd\_data\_in[7:0] and op\_id\_in[7:0] valid then done\_op\_id[7:0] is received in a future cycle and is equal to the input.
15. If wr\_rd\_s\_out HIGH, add\_out[7:0] and sel\_en\_out[4:0] valid then in the next cycle wr\_data\_out[7:0] valid and ack\_in[7:0] is 1 bit HIGH.
16. If FIFO full ready\_out LOW and enable\_in HIGH, next cycle valid\_in HIGH, wr\_rd\_op HIGH, addr\_in[7:0], rd\_data\_in[7:0] and op\_id\_in[7:0] valid then no modification are expected to happen.
17. If FIFO full ready\_out\_HIGH, wr\_rd\_s\_out HIGH, add\_out[7:0] and sel\_en\_out[4:0] valid then in the next cycle ready\_out LOW.
18. If FIFO almost full HIGH (after FIFO\_size – 1 write transactions) ready\_out and enable\_in HIGH, next cycle valid\_in HIGH, wr\_rd\_op HIGH, addr\_in[7:0], rd\_data\_in[7:0] and op\_id\_in[7:0] valid then next cycle ready\_out LOW.
19. If FIFO empty (no transactions) then wr\_data\_out[7:0] will continuously be 0.
20. If addr\_in[7:5] = [5,7] or addr\_in[4:0] = [4,31] and write or read transactionon input is initiated then no modification are expected to happen.

|  |  |  |
| --- | --- | --- |
| Nr. | Name | Description |
| 1 | Reset case | If reset then in the next cycle all output signals have to be 0/LOW. |
| 2 | Read case on input | If enable\_in HIGH, in the next cycle wr\_rd\_op LOW, valid\_in HIGH, addr\_in[7:0] and op\_id\_in[7:0] valid then done\_op\_id[7:0] and rd\_data\_out[7:0] are received in the same future cycle and equal the input. |
| 3 | Read case on output | If wr\_rd\_s\_out LOW, add\_out[7:0] and sel\_en\_out[4:0] valid then in the next cycle rd\_data\_in[7:0] valid and ack\_in[7:0] is 1 bit HIGH. |
| 4 | Write case on input | If enable\_in HIGH, next cycle valid\_in HIGH, wr\_rd\_op HIGH, addr\_in[7:0], rd\_data\_in[7:0] and op\_id\_in[7:0] valid then done\_op\_id[7:0] is received in a future cycle and is equal to the input. |
| 5 | Write case on output | If wr\_rd\_s\_out HIGH, add\_out[7:0] and sel\_en\_out[4:0] valid then in the next cycle wr\_data\_out[7:0] valid and ack\_in[7:0] is 1 bit HIGH. |
| 6 | FIFO full and receives input write | If FIFO full ready\_out LOW and enable\_in HIGH, next cycle valid\_in HIGH, wr\_rd\_op HIGH, addr\_in[7:0], rd\_data\_in[7:0] and op\_id\_in[7:0] valid then no modification are expected to happen. |
| 7 | FIFO full and receives output write | If FIFO full ready\_out\_HIGH, wr\_rd\_s\_out HIGH, add\_out[7:0] and sel\_en\_out[4:0] valid then in the next cycle ready\_out LOW. |
| 8 | FIFO almost full and receives input write | If FIFO almost full HIGH (after FIFO\_size – 1 write transactions) ready\_out and enable\_in HIGH, next cycle valid\_in HIGH, wr\_rd\_op HIGH, addr\_in[7:0], rd\_data\_in[7:0] and op\_id\_in[7:0] valid then next cycle ready\_out LOW. |
| 9 | FIFO empty and receives output write | If FIFO empty then wr\_data\_out[7:0] will continuously be 0. |
| 10 | Address out of bounds | If addr\_in[7:5] = [5,7] or addr\_in[4:0] = [4,31] and write or read transactionon input is initiated then no modification are expected to happen. |

Approaches:

1. Input passive agent; Output passive agent. Randomize everything from testbench.
2. Input active agent; Output active agent. Data is completely random. The input agent ignores the ready\_out signal and the output agent sends random rd\_data\_in and ack (it sends transactions regardless of their values).
3. Input active agent; Output reactive agent. The input agent ignores the ready\_out signal (it sends transactions regardless of its value).
4. Input reactive agent; Output reactive agent. Data is sent according to the specification. UVM env can’t work with only reactive agents.

# Testbench



# Interface

## Input interface

Table 1: Input interface signals

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| NAME | TYPE | SIZE | DESIGN | DRIVER | MONITOR |
| clk | bit | 1 | input | @ | @ |
| rst\_n | bit | 1 | input | - | - |
| enable\_in | bit | 1 | input | output | input |
| wr\_rd\_op | bit | 1 | input | output | input |
| valid\_in | bit | 1 | input | output | input |
| addr\_in | bit | 8 | input | output | input |
| op\_id\_in | bit | 8 | input | output | input |
| wr\_data\_in | bit | 8 | input | output | input |
| ready\_out | bit | 1 | output | - | input |
| rd\_data\_out | bit | 8 | output | - | input |
| done\_op\_id | bit | 8 | output | - | input |

## Output interface

Table 2: Output interface signals

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| NAME | TYPE | SIZE | DESGIN | DRIVER | MONITOR |
| clk | bit | 1 | input | @ | @ |
| rst\_n | bit | 1 | input | - | - |
| rd\_data\_in | bit | 8 | input | output | input |
| ack\_in | bit | 5 | input | output | input |
| sel\_en\_out | bit | 5 | output | - | input |
| wr\_rd\_d\_out | bit | 1 | output | - | input |
| addr\_out | bit | 8 | output | - | input |
| wr\_data\_out | bit | 8 | output | - | input |

# Test

## Item

There are 2 item types: input and output.

Every item has to have the following functions:

* bool check() => returns 1 if item is structural correct, else returns 0
* void copy(item t) => replaces the current item with the given item
* bool compare(item t) => compare the current item with the given item; returns 1 if equal, else returns 0
* void default() => resets all values to default

### Input item

Table 3: Input item

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| NAME | TYPE | SIZE | RANDOM | CONSTRAIN |
| enable\_in | bit | 1 | YES |  |
| wr\_rd\_op | bit | 1 | YES |  |
| valid\_in | bit | 1 | YES |  |
| addr\_in | bit | 8 | YES |  |
| op\_id\_in | bit | 8 | YES |  |
| wr\_data\_in | bit | 8 | YES |  |
| ready\_out | bit | 1 | NO |  |
| rd\_data\_out | bit | 8 | NO |  |
| done\_op\_id | bit | 8 | NO |  |

### Output item

Table 4: Output item

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| NAME | TYPE | SIZE | RANDOM | CONSTRAIN |
| rd\_data\_in | bit | 8 | YES |  |
| ack\_in | bit | 5 | NO |  |
| sel\_en\_out | bit | 5 | NO |  |
| wr\_rd\_d\_out | bit | 1 | NO |  |
| addr\_out | bit | 8 | NO |  |
| wr\_data\_out | bit | 8 | NO |  |

## Sequence

### Input sequence

### Output sequence

### Virtual sequence

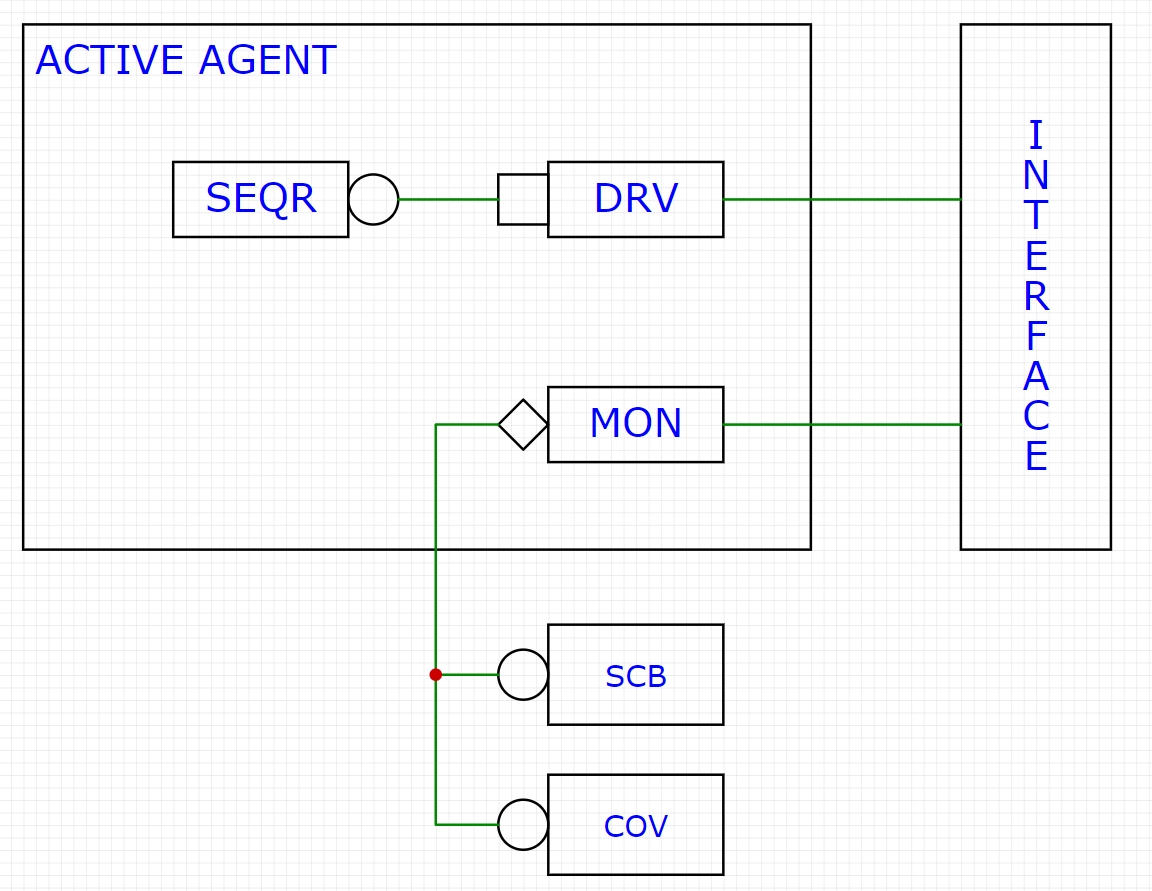
Forever get monitor response and create new request sequence.

Request ack\_in = Response sel\_en\_out

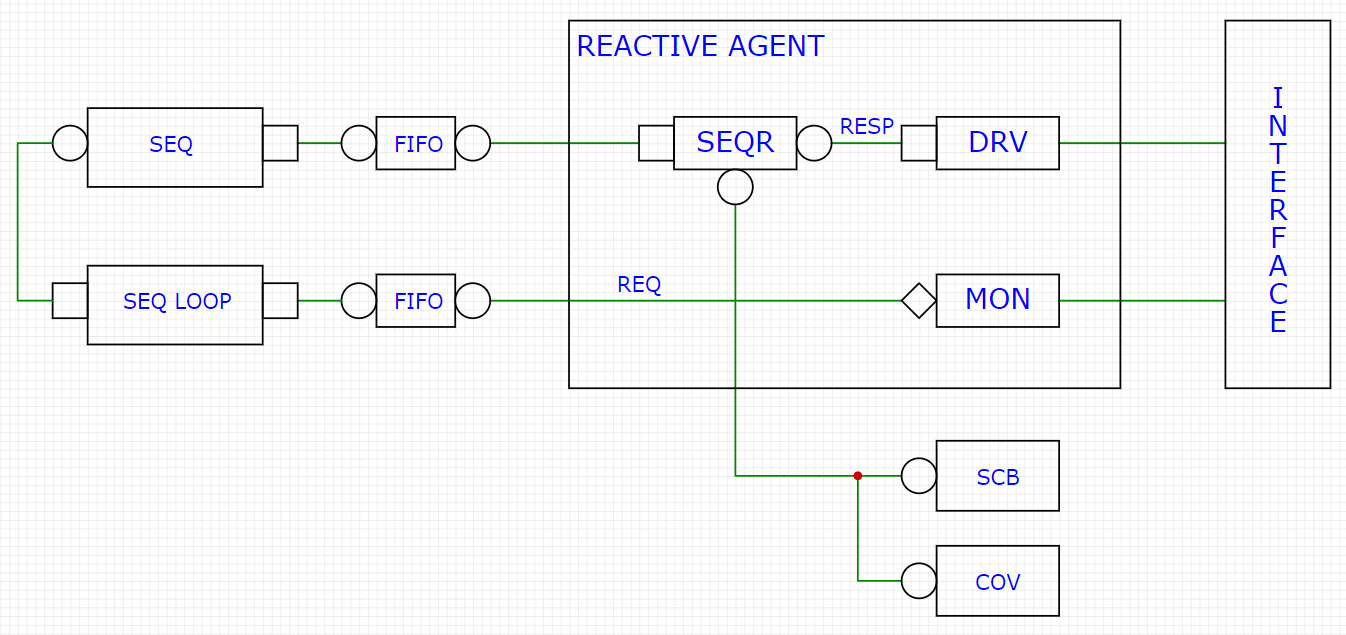
# Environment

## Virtual sequencer

## Active Agent



## Reactive Agent



## Scoreboard

## Coverage

Separate file.

# Display