SPECIFICATION

Contents

[DUT 4](#_Toc150613403)

[PHASES 5](#_Toc150613404)

[BUILD 5](#_Toc150613405)

[CONNECT 5](#_Toc150613406)

[START OF SIMULATION. 6](#_Toc150613407)

[RUN 6](#_Toc150613408)

[RESET 6](#_Toc150613409)

[CONFIGURE 6](#_Toc150613410)

[MAIN 6](#_Toc150613411)

[REPORT 7](#_Toc150613412)

[TESTBENCH 7](#_Toc150613413)

[TEST 8](#_Toc150613414)

[ENVIRONMENT 9](#_Toc150613415)

[INTERFACE 10](#_Toc150613416)

[PORT INTERFACE 10](#_Toc150613417)

[CONTROL INTERFACE 11](#_Toc150613418)

[MEMORY INTERFACE 11](#_Toc150613419)

[RESET INTERFACE 11](#_Toc150613420)

[AGENT 12](#_Toc150613421)

[PROACTIVE AGENT 13](#_Toc150613422)

[REACTIVE AGENT 14](#_Toc150613423)

[CONFIGURATION OBJECT 15](#_Toc150613424)

[DRIVER 16](#_Toc150613425)

[MONITOR 17](#_Toc150613426)

[ITEM 18](#_Toc150613427)

[DATA PACKET ITEM 18](#_Toc150613428)

[CONTROL ITEM 19](#_Toc150613429)

[MEMORY ITEM 19](#_Toc150613430)

[PORT ITEM 19](#_Toc150613431)

[RESET ITEM 20](#_Toc150613432)

[SEQUENCE 20](#_Toc150613433)

[CONTROL SEQEUNCE 20](#_Toc150613434)

[MEMORY SEQEUNCE 20](#_Toc150613435)

[PORT SEQEUNCE 20](#_Toc150613436)

[RESET SEQEUNCE 21](#_Toc150613437)

[SCOREBOARD 21](#_Toc150613438)

[COVERAGE 21](#_Toc150613439)

[PORT COVERGROUP 22](#_Toc150613440)

[MEMORY COVERGROUP 22](#_Toc150613441)

[EVENT COVERGROUP 23](#_Toc150613442)

[DATA COVERGROUP 23](#_Toc150613443)

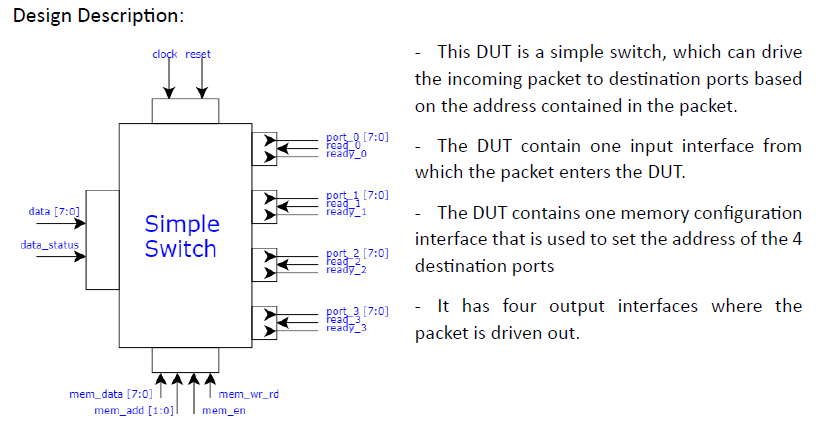
[PORT COVERAGE 24](#_Toc150613444)

[PACKAGE 24](#_Toc150613445)

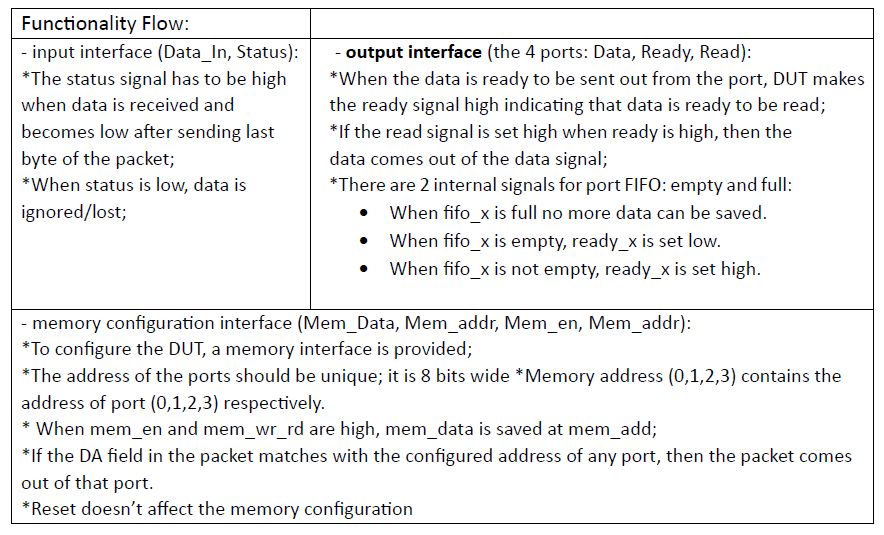
[VERBOSITY 25](#_Toc150613446)

# 

# DUT







# PHASES

## BUILD

|  |  |
| --- | --- |
| Component | Description |
| Test | A new environment configuration object is created and assigned to the environment. The environment is created. The sequences are created. |
| Environment | A new environment configuration object is created and takes the assigned configuration object from the database. Multiple agent configuration objects are created and assigned to the agents. The agents are created. The scoreboard is created. The coverage is created. |
| Agent control | A new agent configuration object is created and takes the assigned configuration object from the database. If the agent is active, a sequencer and a driver are created. A monitor is created. |
| Agent reset | A new agent configuration object is created and takes the assigned configuration object from the database. If the agent is active, a sequencer and a driver are created. A monitor is created. |
| Agent memory | A new agent configuration object is created and takes the assigned configuration object from the database. If the agent is active, a sequencer and a driver are created. A monitor is created. |
| Agent port | A new agent configuration object is created and takes the assigned configuration object from the database. If the agent is active, a sequencer and a driver are created. A monitor is created. |
| Driver control | The interface is taken from the database. |
| Driver reset | The interface is taken from the database. |
| Driver memory | The interface is taken from the database. |
| Driver port | The interface is taken from the database. |
| Monitor control | The interface is taken from the database. |
| Monitor reset | The interface is taken from the database. |
| Monitor memory | The interface is taken from the database. |
| Monitor port | The interface is taken from the database. |

## CONNECT

|  |  |
| --- | --- |
| Component | Description |
| Agent control | If the agent is active the driver is connected to the monitor. The monitor analysis port is connected to the agent analysis port. |
| Agent reset | If the agent is active the driver is connected to the monitor. The monitor analysis port is connected to the agent analysis port. |
| Agent memory | If the agent is active the driver is connected to the monitor. The monitor analysis port is connected to the agent analysis port. |
| Agent port | If the agent is active the driver is connected to the monitor. The monitor analysis port is connected to the agent analysis port. |
| Environment | The virtual sequencer sequencers are connected to the port agents’ sequencer. All monitors are connected to the scoreboard and coverage. The port agents’ monitors are connected to the port coverage. |

## START OF SIMULATION.

The topology is printed from test.

## RUN

### RESET

|  |  |
| --- | --- |
| Component | Description |
| Driver control | All data is set to idle:   * data: 8’h00 * data\_status: 1’b0 |
| Driver reset | Reset is activated and deactivated after a clock. |
| Driver memory | All data is set to idle:   * mem\_data: 8’h00 * mem\_add: 2’b00 * mem\_en: 1’b0 * mem\_rd\_wr: 1’b0 |
| Driver port | All data is set to idle:   * read: 1’b0 |

### CONFIGURE

|  |  |
| --- | --- |
| Component | Description |
| Driver memory | Every memory address is set to have a set value received from test. |

### MAIN

|  |  |
| --- | --- |
| Component | Description |
| Test | The sequences are started. |
| Driver control | The driver gets the next item from the sequencer and sends it to the interface. |
| Driver reset | The driver gets the next item from the sequencer and sends it to the interface. |
| Driver memory | The driver gets the next item from the sequencer and sends it to the interface. |
| Driver port | The driver gets the next item from the sequencer and sends it to the interface. |
| Monitor control | The monitor gets the item from the interface. If the item is the same as the previous item then it is ignored. If the item is different it is printed and sent to the analysis port. |
| Monitor reset | The monitor gets the item from the interface. If the item is the same as the previous item then it is ignored. If the item is different it is printed and sent to the analysis port. |
| Monitor memory | The monitor gets the item from the interface. If the item is the same as the previous item then it is ignored. If the item is different it is printed and sent to the analysis port. |
| Monitor port | The monitor gets the item from the interface. If the item is the same as the previous item then it is ignored. If the item is different it is printed and sent to the analysis port. |

## REPORT

|  |  |
| --- | --- |
| Component | Description |
| Scoreboard | The following values are printed:   * miss / total * match % * Number of packets * Number of dropped packets * Number of packets received and sent by each port |
| Coverage | The coverage for each cover group is printed. |
| Environment | The coverage for each individual port is printed. |

# TESTBENCH

The testbench connects the environment and the DUT. The next steps are followed:

* declare the clock signal
* generate the clock
* create an instance of the input interface
* create an instance of the memory interface
* create four instances of the output interface
* instantiate the DUT
* set the interfaces for each agent
* enable the dump file
* run the test

# TEST

The test class extends the uvm\_test class. The next steps are followed:

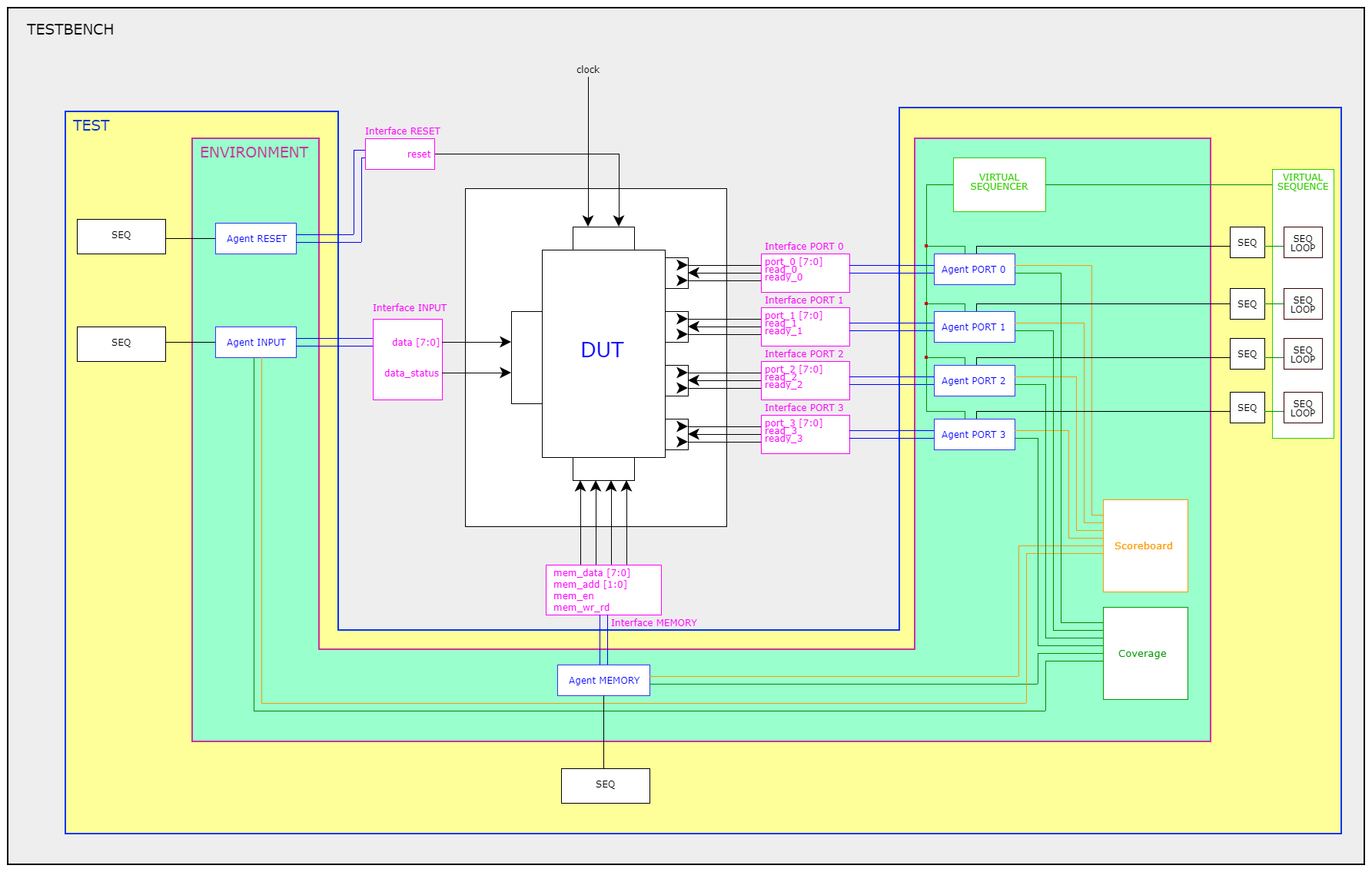
* declare the environment
* declare the sequences
* create the environment in the build phase
* configure the environment in the build phase
* create the sequences
* print the topology in the start of simulation phase
* start the sequences in the main phase

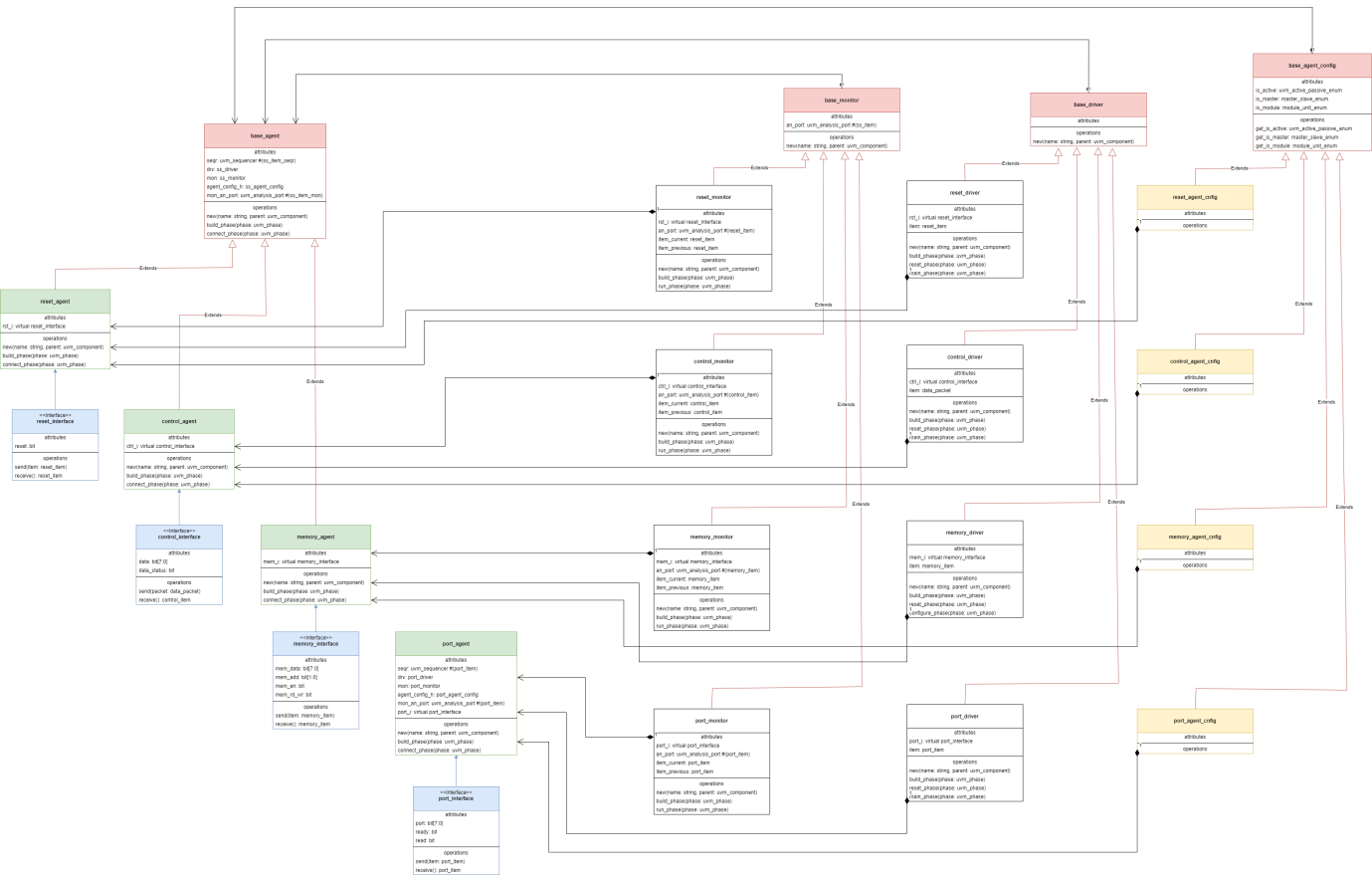
There are 12 test cases and the smoke test.

From these 12 test cases only 10 can be verified for the current design. Test no. 3 and 10 require changes to the design.

* test\_no\_1: Memory undefined configuration
* test\_no\_2: Deactivate status in middle of transaction
* test\_no\_3: Read data in parallel
* test\_no\_4: Memory not unique configuration
* test\_no\_5: Packet destination address does not match
* test\_no\_6: Different length packets
* test\_no\_7: Back to back packets
* test\_no\_8: Activate reset in middle of transaction
* test\_no\_9: Identical packet fields
* test\_no\_10: Change port address in middle of transaction
* test\_no\_11: Port memory full
* test\_no\_12: Read cases

# ENVIRONMENT





The environment class supports the testbench structure. The next steps are followed:

* declare the agents, interfaces, coverage and scoreboard
* create the agents, interfaces, coverage and scoreboard
* connect the analysis ports of the agents to the scoreboard and coverage

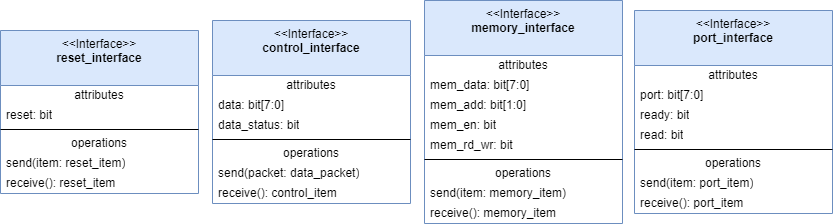
# INTERFACE

There are 4 different interfaces connected to the DUT:

* INPUT INTERFACE (CONTROL INTERFACE)
* RESET INTERFACE
* MEMORY INTERFACE
* OUTPUT INTERFACE (PORT INTERFACE)

Each interface has:

* signals
* a driver clocking block
* a monitor clocking block
* a send task - send(item\_type item) – which drives the signals
* a receive function - receive(item\_type item) – which monitors the signals



## PORT INTERFACE

The output interface is connected to the output ports of the DUT. This interface is instantiated four times. The next steps are followed:

* declare the signals
  + port – 8 bits
  + read – 1 bit
  + ready – 1 bit
* declare the driver clocking block
  + input port
  + output read
  + input ready
* declare the monitor clocking block
  + input port
  + input read
  + input ready

## CONTROL INTERFACE

The input interface is connected to the input port of the DUT and the reset signal. The next steps are followed:

* declare the signals
  + data – 8 bits
  + data\_status – 1 bit
* declare the driver clocking block
  + output data
  + output data\_status
* declare the monitor clocking block
  + input data
  + input data\_status

## MEMORY INTERFACE

The output interface is connected to the memory port of the DUT. The next steps are followed:

* declare the signals
  + mem\_data – 8 bits
  + mem\_addr – 2 bits
  + mem\_en – 1 bit
  + mem\_wr\_rd – 1 bit
* declare the driver clocking block
  + output mem\_data
  + output mem\_addr
  + output mem\_en
  + output mem\_wr\_rd
* declare the monitor clocking block
  + input mem\_data
  + input mem\_addr
  + input mem\_en
  + input mem\_wr\_rd

## RESET INTERFACE

The reset interface is connected to the reset signal of the DUT. The next steps are followed:

* declare the signals
  + reset – 1 bit
* declare the driver clocking block
  + output reset
* declare the monitor clocking block
  + input reset

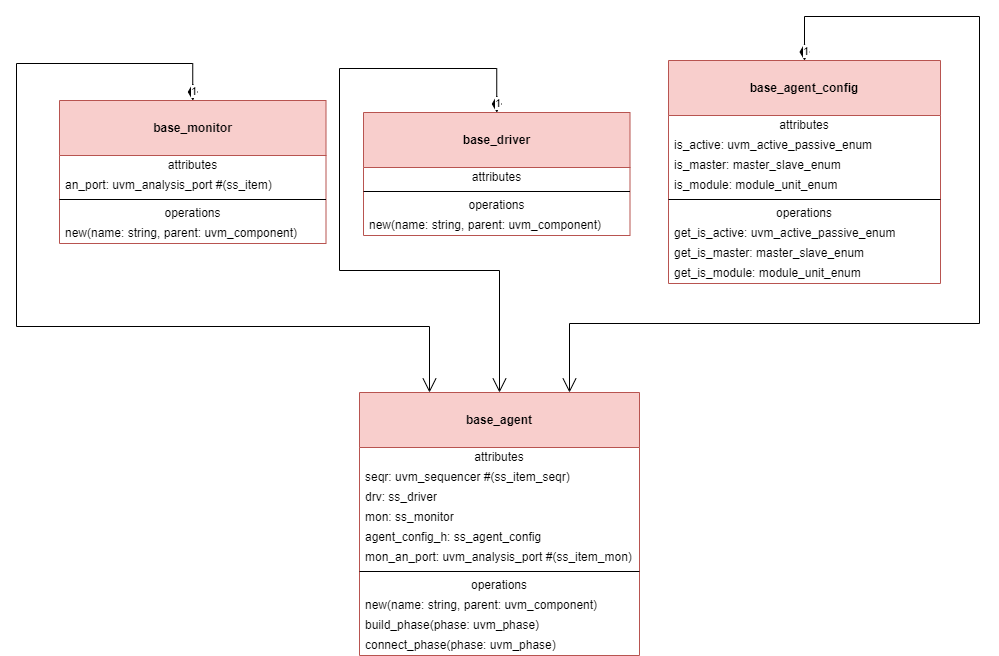
# AGENT

The base\_agent class extends the uvm\_agent class. The next steps are followed:

* declare the sequencer, monitor and driver
* if the agent is active: create the sequencer, monitor and driver in the build phase, else if the agent is passive create the monitor in the build phase
* if the agent is active connect the sequencer and the driver in the connect phase

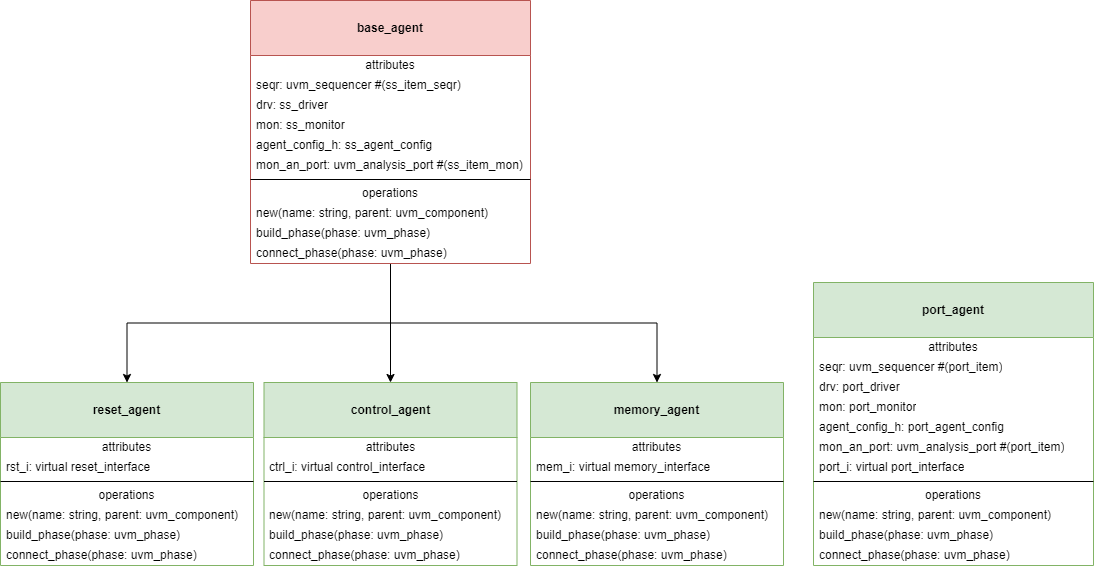
The base\_agent class has the following parameters:

* string name = "base\_agent"
* type ss\_agent\_config = base\_agent\_config
* type ss\_item\_seqr = uvm\_sequence\_item,
* type ss\_item\_mon = uvm\_sequence\_item
* type ss\_driver = base\_driver
* type ss\_monitor = base\_monitor

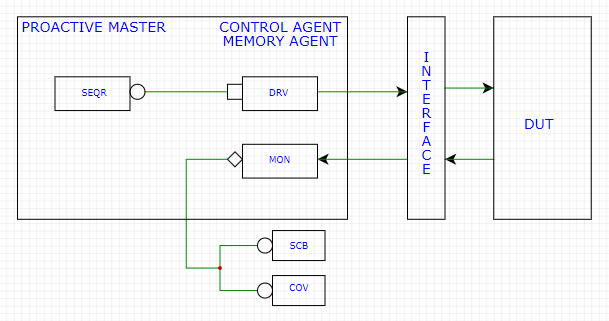


The base\_agent is the base class used for the other agents:

* RESET AGENT
* CONTROL AGENT
* MEMORY AGENT

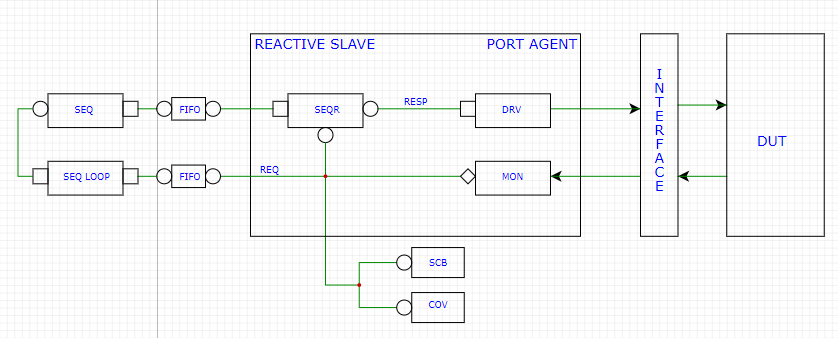


## PROACTIVE AGENT



The RESET, CONTROL and MEMORY AGENTS are proactive agents. They have a sequencer connected to a driver and a monitor connected to the scoreboard and coverage.

## REACTIVE AGENT



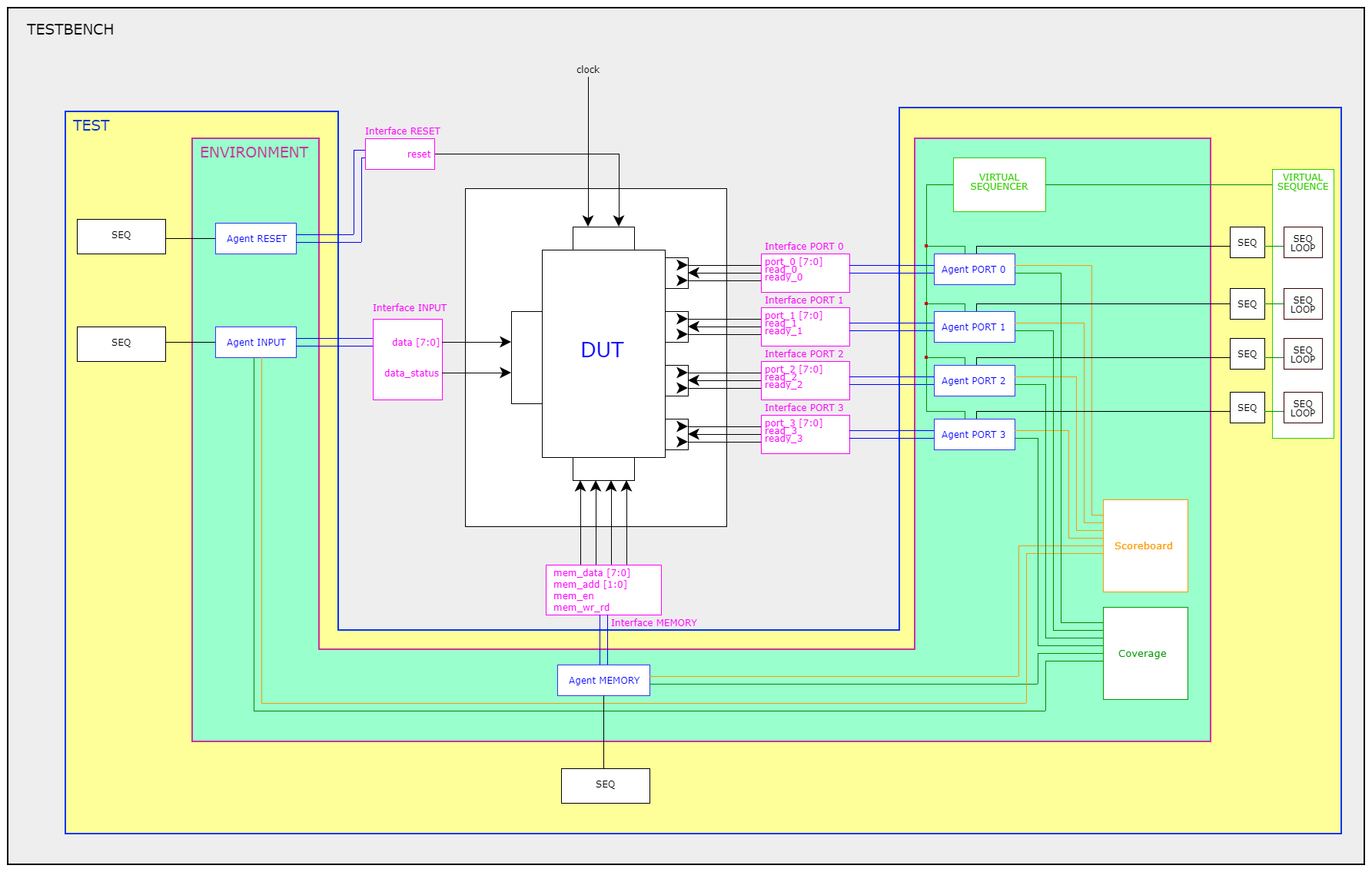
The PORT AGENT is a reactive agent. The sequencer is connected to the driver, the monitor is connected to the scoreboard and coverage.

Every PORT AGENT has a PORT SEQUENCER, which contains an analysis port and a fifo.

The VIRTUAL SEQUENCER contains the 4 port sequencers. In the environment, the virtual sequencer sequencers are connected to the sequencers in the agents.

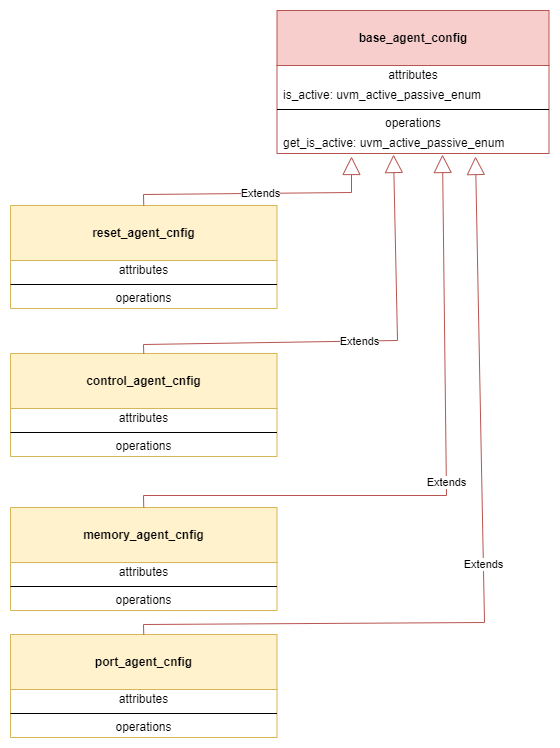
The VIRTUAL SEQUENCE contains 4 sequences which start the reactive sequence for the reactive agent. Each loop checks for the port ready signal. The virtual sequence is then created in the test.

The sequence sent to the port contains one port item with the read bit set to 1’b0 or 1’b1 after the ready bit received from the loop.



# CONFIGURATION OBJECT

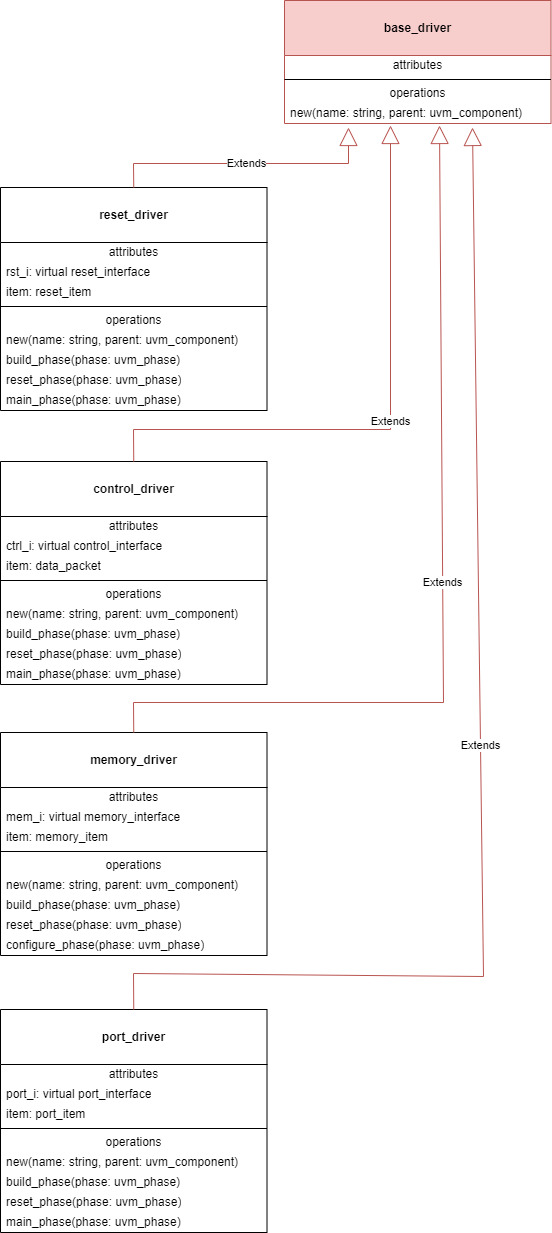
Each agent has a configuration object which extends the base configuration object.



# DRIVER

The base\_driver class extends the uvm\_driver class. It is the base class for:

* RESET DRIVER
* CONTROL DRIVER
* MEMORY DRIVER
* PORT DRIVER

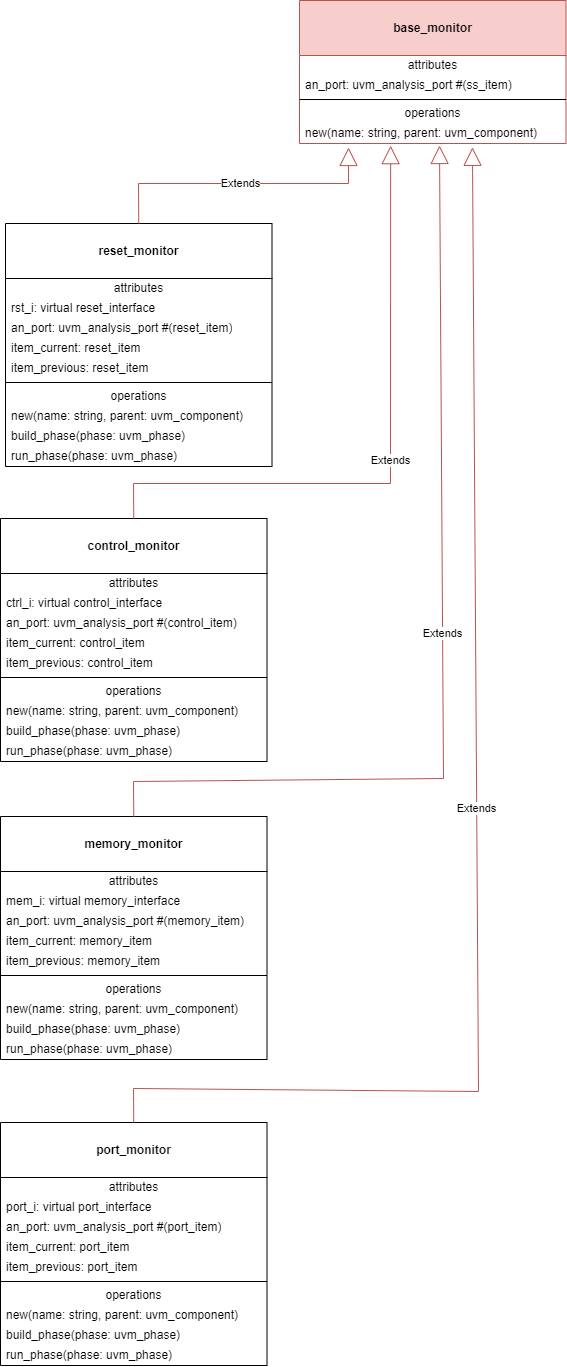


Each driver resets the signals during the reset phase. Each driver sends data to the DUT during the main phase.

# MONITOR

The base\_monitor class extends the uvm\_monitor class. It is the base class for:

* RESET MONITOR
* CONTROL MONITOR
* MEMORY MONITOR
* PORT MONITOR



Each monitor receives the data from the interface and sends it to the analysis port. For an item to be sent forward it has to be valid:

|  |  |
| --- | --- |
| MONITOR | CONDITION |
| MEMORY | The previous item has to be different from the current item. |
| PORT | Ready has to be active or the previous item has to be different from the current item. |
| RESET | The previous item has to be different from the current item. |
| CONTROL | Data status has to be active or the previous item has to be different from the current item. |

# ITEM

There are 5 item types:

* DATA PACKET
* RESET ITEM
* CONTROL ITEM
* MEMORY ITEM
* PORT ITEM

The data packet describes the data packet described by the DUT specifications. The other items are each correlated to a different interface.

Each item has:

* convert2string – which converts the data into a string
* compare – which compares this item with the given one
* copy – which returns a deep copy of the given item

## DATA PACKET ITEM

This packet has the following structure:

* da = destination address – 1 byte
* sa = source address – 1 byte
* length = payload length – 1 byte
* data = payload data – 0-255 bytes
* data\_status – 3-259 bytes

It also has the following control variables:

|  |  |  |  |
| --- | --- | --- | --- |
| NAME | TYPE | DEFAULT | MEANING |
| delay | Integer | 5 | the delay after the package is sent to the DUT |
| max\_length | Integer | 255 | payload maximum length |
| min\_length | Integer | 0 | payload minimum length |
| random\_DA | Enum | PREDEFINED | if it is set to PREDEFINED DA can only take memory data values; otherwise DA is random |

Data packets has the following functions:

* function void set\_parameters(int min\_length, int max\_length, bit [7:0] memory\_data[4] , enum random\_DA) - sets the parameters from the sequence
* function void set\_status\_low(int position) - sets status low at a certain position
* function void set\_all(bit[7:0] da, bit[7:0] sa, bit[7:0] length, bit[7:0] pay) - sets the entire data packet – no randomization
* function bit check() - check if the packet length is equal to the payload size

## CONTROL ITEM

This item is used by the control agent and has the following structure:

* data – 1 byte
* data\_status – 1 bit

## MEMORY ITEM

This item is used by the memory agent and has the following structure:

* mem\_data – 1 byte
* mem\_addr – 2 bits
* mem\_en – 1 bit
* mem\_wr\_rd – 1 bit

Any signal can be overwritten using the set functions:

* function void set\_data(logic [7:0] mem\_data);
* function void set\_address(bit [1:0] mem\_add);
* function void set\_enable(bit mem\_en, bit mem\_rd\_wr);
* function void set\_item(logic [7:0] mem\_data, bit [1:0] mem\_add, bit mem\_en, bit mem\_rd\_wr);

## PORT ITEM

This item is used by the port agent and has the following structure:

* port – 1 byte
* ready – 1 bit
* read – 1 bit

It has the bandwidth variable, which sets the probability of a read to be active.

## RESET ITEM

This item is used by the reset agent and has the following structure:

* reset – 1 bit

# SEQUENCE

There are 4 different sequences, one for each port:

* RESET SEQUENCE
* CONTROL SEQUENCE
* MEMORY SEQUENCE
* PORT SEQUENCE

## CONTROL SEQEUNCE

The control sequence creates a data packet and sets its delay. It has the following parameters:

* nr\_items – number of configuration created
* max\_length – packet maximum length
* min\_length – packet minimum length
* duration – the duration of the status low special case
* position – the position of the status low special case
* enable\_status\_low – enable the status low special case
* random\_DA - if it is set to PREDEFINED DA can only take memory data values; otherwise DA is random
* no\_delay – decides if there is no delay after package
* no\_random – decides if the configutation is random or not

## MEMORY SEQEUNCE

The memory sequence creates a pair of items, one active memory configuration and a disable item. It has the following parameters:

* nr\_items – number of configuration created
* addr – the address of the configuration
* no\_random – decides if the configutation is random or not

## PORT SEQEUNCE

The port sequence creates an item. The item is randomized if ready is active, otherwise the items’ read signal is set to 0.

## RESET SEQEUNCE

The reset sequence creates a pair of items, one active reset and one inactive reset.

# SCOREBOARD

The scoreboard contains 7 analysis imports, one for each agent.

In the build phase the imports are created and the port queues, memory arrays, items and report variables needed for verification are instantiated.

In the report phase the miss rate is displayed.

Each import has its own write function:

* PORT : if data is valid than data is compared with the respective port queue
* CONTROL : if DA matches a memory than the port queue with the respective memory address is filled as long as data status is active
* MEMORY : if memory is enabled than the memory is saved in the memory array
* RESET : if reset is active than the port queues are deleted

# COVERAGE

The coverage contains 5 different coverage groups:

* MEMORY COVERGROUP
* CONTROL COVERGROUP
* PORT COVERGROUP
* DATA COVERGROUP
* EVENT COVERGROUP

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Nr. | Covergroup | Interfaces | Nr. Cases | Nr. cvp | Nr. Cross |
| 1 | cvg\_pachet | control | 18 | 18 | 0 |
| 2 | cvg\_port | port 0 | 15 | 11 | 4 |
| 3 | port 1 | 15 | 11 | 4 |
| 4 | port 2 | 15 | 11 | 4 |
| 5 | port 3 | 15 | 11 | 4 |
| 6 | cvg\_mem | memory | 32 | 12 | 20 |
| 7 | cvg\_event | control, memory, port 0-3 | 414 | 22 | 392 |
|  |  |  | 524 | 96 | 428 |

## PORT COVERGROUP

Each port has to sample the following data:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| covergroup | coverpoint | trigger | variable | bins | nr bins | code |
| cvg\_port | cvp\_data |  | data | bin | 7 | 0, [1 : 84], 85, [86 : 169], 170, [171 : 254], 255 |
| cvp\_read |  | read | bin | 2 | 0, 1 |
| cvp\_ready |  | ready | bin | 2 | 0, 1 |
|  |  |  |  |  | 11 |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| covergroup | cross coverpoint | trigger | variable coverpoint | bins | nr bins | code |
| cvg\_port | cross\_receive |  | cvp\_read | bin | 2x2 = 4 |  |
| cvp\_ready |
|  |  |  |  |  | 4 |  |

There are 60 bins in total.

## MEMORY COVERGROUP

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| covergroup | coverpoint | trigger | variable | bins | nr bins | code |
| cvg\_mem | cvp\_mem\_rd\_wr |  | mem\_rd\_wr | bin | 2 | 0, 1 |
| cvp\_mem\_en |  | mem\_en | bin | 2 | 0, 1 |
| cvp\_mem\_add |  | mem\_add | bin | 4 | 0, 1, 2, 3 |
| cvp\_mem\_data |  | mem\_data | bin | 4 | 0, 85, 170, 255 |
|  |  |  |  |  | 12 |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| covergroup | cross coverpoint | trigger | variable coverpoint | bins | nr bins | code |
| cvg\_mem | cross\_save |  | cvp\_mem\_rd\_wr | bin | 2x2 = 4 |  |
| cvp\_mem\_en |
| cross\_data | mem\_en & mem\_rd\_wr | cvp\_mem\_add | bin | 4x4 = 16 |  |
| cvp\_mem\_data |
|  |  |  |  |  | 20 |  |

There are 32 bins in total.

## EVENT COVERGROUP

This covergroup covers the special event that can occur:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| covergroup | coverpoint | trigger | variable | bins | nr bins | code |
| cvg\_event | cvp\_mem\_rd\_wr |  | mem\_rd\_wr | bin | 2 | 0, 1 |
| cvp\_mem\_en |  | mem\_en | bin | 2 | 0, 1 |
| 4xcvp\_read |  | read | bin | 4x2 | 0, 1 |
| 4xcvp\_ready |  | ready | bin | 4x2 | 0, 1 |
| cvp\_status |  | data\_status | bin | 2 | 0, 1 |
|  |  |  |  |  | 22 |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| covergroup | cross coverpoint | trigger | variable coverpoint | bins | nr bins | code |
| cvg\_event | cross\_control |  | cvp\_mem\_rd\_wr | bin | 2x2x2 = 8 |  |
| cvp\_mem\_en |
| cvp\_status |
| cross\_port |  | cvp\_mem\_rd\_wr | bin | 2x2x4x2x 4x2 = 256 |  |
| cvp\_mem\_en |
| cvp\_read |
| cvp\_ready |
| cross\_rd\_wr |  | cvp\_read | bin | 2x4x2x4x2 = 128 |  |
| cvp\_ready |
| data\_status |
|  |  |  |  |  | 392 |  |

There are 414 bins in total.

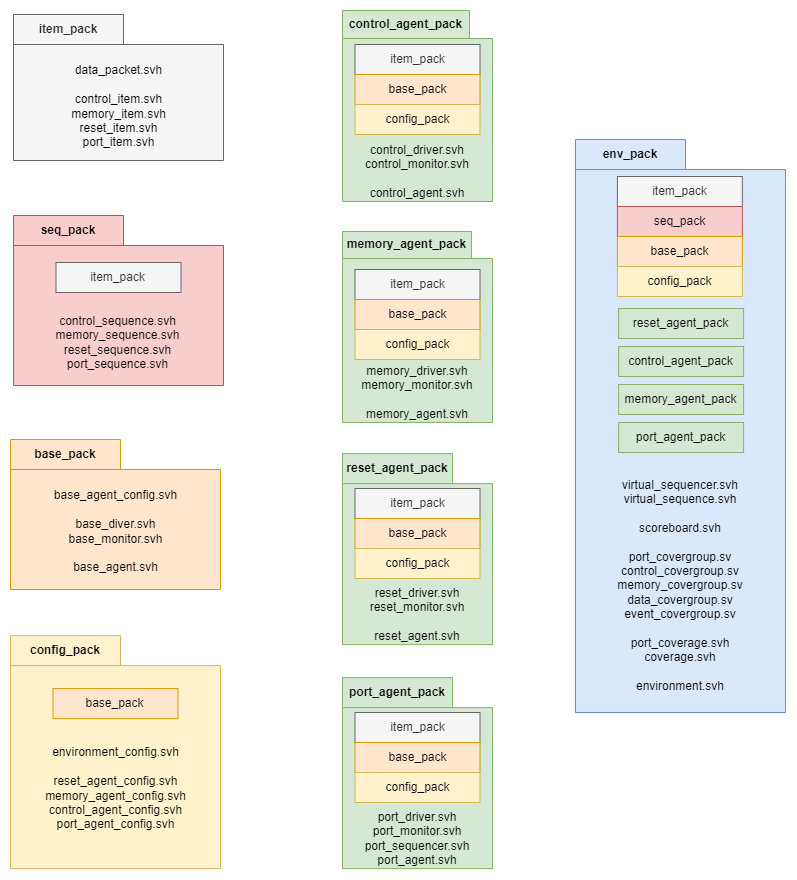
## DATA COVERGROUP

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| covergroup | coverpoint | trigger | variable | bins | nr bins | code |
| cvg\_pachet | cvp\_da |  | da | bin | 7 | 0, [1 : 84], 85, [86 : 169], 170, [171 : 254], 255 |
| cvp\_sa |  | sa | bin | 7 | 0, [1 : 84], 85, [86 : 169], 170, [171 : 254], 255 |
| cvp\_length |  | length | bin | 3 | 3, [4 : 254], 255 |
| cvp\_data |  | data | bin | 1 | [0: 255] |
|  |  |  |  |  | 18 |  |

# PORT COVERAGE

The port coverage class contains one port cover group and is used to obtain the individual port coverage.

## PACKAGE



# VERBOSITY

|  |  |  |
| --- | --- | --- |
| **VERBOSITY** | **DESCRIPTION** | **COMPONENT** |
| UVM\_NONE |  |  |
| UVM\_LOW | Reset driver  Reset monitor  Coverage  Scoreboard  Miss scoreboard | Reset driver  Reset monitor  Coverage  Scoreboard |
| UVM\_MEDIUM | Memory monitor  Start / Finish sequence | Memory monitor  All sequences without Virtual Sequences |
| UVM\_HIGH | Start drive  Finish drive | All |
| UVM\_FULL | Control monitor  Match scoreboard | Control monitor  Scoreboard |
| UVM\_DEBUG | Enter phase  Exit phase | All |
| UVM\_FATAL | Failed to get configuration object  Failed to get interface | All |