1. Describe the below methods in UVM Transaction with syntax & an example:
   1. copy

The copy method is used to copy the data of one object to another. It is typically used to duplicate the contents of one transaction object into another of the same type.

Syntax: void copy(input uvm\_object rhs);

Example:

my\_transaction t1, t2;

t1.some\_field = 5;

t2.copy(t1); // Copy t1's data into t2

* 1. do\_copy

The do\_copy method is a specialised method (overloaded in a transaction class) that handles copying the contents of one object to another. It is generally used internally to define the actual copying behaviour.

Syntax: virtual function void do\_copy(input uvm\_object rhs);

Example:

class my\_transaction extends uvm\_transaction;

int some\_data;

function void do\_copy(input uvm\_object rhs);

super.do\_copy(rhs); // Call the base class's do\_copy

my\_transaction t = my\_transaction(rhs);

some\_data = t.some\_data;

endfunction

endclass

* 1. compare

The compare method compares the current object with another to check if their contents are identical.

Syntax: virtual function int compare(input uvm\_object rhs);

Example:

my\_transaction t1, t2;

t1.some\_field = 5;

t2.some\_field = 5;

if (t1.compare(t2) == 0) {

$display("Transactions are identical");

}

* 1. convert2string

The convert2string method is used to return a string representation of the object's contents.

Syntax: virtual function string convert2string();

Example: my\_transaction t;

t.some\_field = 5;

string str = t.convert2string(); // Converts transaction to a string

* 1. print

The print method is used to print the object's contents to the simulation log.

Syntax: virtual function void print(string prefix = "");

Example:

my\_transaction t;

t.some\_field = 5;

t.print(); // Prints the transaction details

* 1. sprint

The sprint method is used to return a string representation of the object’s contents. It is similar to convert2string, but sprint is generally used in UVM for formatted output.

Syntax: virtual function string sprint(string prefix = "");

Example: my\_transaction t;

t.some\_field = 10;

string str = t.sprint(); // Returns a string formatted from the transaction's data

* 1. record

The record method is used to record the contents of the object. It is generally used for logging purposes or capturing the state of the object.

Syntax: virtual function void record();

Example:

my\_transaction t;

t.some\_field = 20;

t.record(); // Records or logs the current state of the transaction

* 1. pack

The pack method is used to serialize an object into a byte stream. It is typically used when sending a transaction over a communication channel (e.g., for logging, communication, or saving data).

Syntax: virtual function void pack(uvm\_packer packer);

Example:

my\_transaction t;

uvm\_packer packer;

t.pack(packer); // Pack the transaction into the packer

* 1. unpack

The unpack method is used to deserialize an object from a byte stream. It is the reverse of pack.

Syntax: virtual function void unpack(uvm\_packer packer);

Example:

my\_transaction t;

uvm\_packer packer;

t.unpack(packer); // Unpack the data from the packer into the transaction

* 1. to\_struct

The to\_struct method converts the UVM object into a SystemVerilog struct. This is useful when you need to work with non-UVM models.

Syntax: virtual function uvm\_object to\_struct();

Example:

my\_transaction t;

uvm\_object struct\_rep = t.to\_struct(); // Converts the transaction to a struct

* 1. from\_struct

The from\_struct method is used to convert a SystemVerilog struct back into a UVM object.

Syntax: virtual function void from\_struct(input uvm\_object struct\_rep);

Example:

my\_transaction t;

uvm\_object struct\_rep;

t.from\_struct(struct\_rep); // Converts the struct back into a UVM transaction

1. Describe the below methods in sequence\_item with syntax and example:
   1. start\_item()

The start\_item() method is called by a sequence to start the process of sending a sequence item to a driver. It reserves space for the item in the transaction queue and makes it ready to be processed by the driver.

Syntax: virtual function uvm\_status\_e start\_item(output uvm\_sequence\_item item);

Example:

class my\_sequence extends uvm\_sequence;

`uvm\_object\_utils(my\_sequence)

virtual task body();

my\_sequence\_item item;

uvm\_status\_e status;

// Create a new sequence item

item = my\_sequence\_item::type\_id::create("item");

// Start the item and check the result

status = start\_item(item);

if (status != UVM\_IS\_OK) begin

`uvm\_error("SEQ\_ERR", "Failed to start the sequence item.")

end

endtask

endclass

* 1. finish\_item()

The finish\_item() method is used to indicate that the sequence item has finished executing and the sequence item is now complete. This method is called after the driver has completed the execution of the sequence item, allowing UVM to release resources or trigger subsequent actions.

Syntax: virtual function void finish\_item(input uvm\_sequence\_item item);

Example:

class my\_sequence extends uvm\_sequence;

`uvm\_object\_utils(my\_sequence)

virtual task body();

my\_sequence\_item item;

// Create and start the sequence item

item = my\_sequence\_item::type\_id::create("item");

start\_item(item);

// After the item is done, finish it

finish\_item(item); // Mark the item as finished

endtask

endclass

* 1. get\_reposnse()

The get\_response() method retrieves the response associated with a sequence item, typically set by the driver or the sequencer during or after the execution of a sequence item. This method is useful to fetch data or status information generated by the driver after processing the item.

Syntax: virtual function uvm\_sequence\_item get\_response();

Example:

class my\_sequence extends uvm\_sequence;

`uvm\_object\_utils(my\_sequence)

virtual task body();

my\_sequence\_item item, response;

// Create and start the sequence item

item = my\_sequence\_item::type\_id::create("item");

start\_item(item);

// After finishing the item, get the response

response = get\_response();

// Do something with the response

if (response != null) begin

`uvm\_info("SEQ\_RESP", "Received response: " + response.convert2string(), UVM\_HIGH)

end

endtask

endclass

1. Describe the below methods in sequence-driver API with syntax and example:
   1. get\_next\_item

The get\_next\_item() method retrieves the next item from the sequence for the driver to execute. It blocks the driver until a new item is available.

Syntax: virtual function uvm\_status\_e get\_next\_item(output uvm\_sequence\_item item);

Example:

class my\_driver extends uvm\_driver;

`uvm\_component\_utils(my\_driver)

virtual task run\_phase(uvm\_phase phase);

my\_sequence\_item item;

// Wait for the next item from the sequence

while (1) begin

get\_next\_item(item); // Retrieve the next item

// Process the item (e.g., drive it to DUT)

// ...

end

endtask

endclass

* 1. try\_next\_item

The try\_next\_item() method attempts to fetch the next sequence item but does not block the driver if no item is available. It returns immediately with a success or failure status.

Syntax: virtual function uvm\_status\_e try\_next\_item(output uvm\_sequence\_item item);

Example:

class my\_driver extends uvm\_driver;

`uvm\_component\_utils(my\_driver)

virtual task run\_phase(uvm\_phase phase);

my\_sequence\_item item;

// Try to fetch the next item without blocking

if (try\_next\_item(item) == UVM\_IS\_OK) begin

// Process the item if available

// ...

end

else begin

// No item available, perform other actions or idle

// ...

end

endtask

endclass

* 1. item\_done

The item\_done() method is called by the driver to indicate that it has finished processing the current item and it is now ready for the next item. This is typically used by the driver to notify the sequencer that it is ready to receive another item.

Syntax: virtual function void item\_done(input uvm\_sequence\_item item);

Example:

class my\_driver extends uvm\_driver;

`uvm\_component\_utils(my\_driver)

virtual task run\_phase(uvm\_phase phase);

my\_sequence\_item item;

// Get the next item

get\_next\_item(item);

// Process the item (e.g., driving signals to DUT)

// ...

// Indicate that the item has been processed

item\_done(item); // Notify the sequencer that item is done

endtask

endclass

* 1. peek

The peek() method retrieves the next sequence item without removing it from the queue. This allows the driver to check what the next item is, but it does not consume it.

Syntax: virtual function uvm\_status\_e peek(output uvm\_sequence\_item item);

Example:

class my\_driver extends uvm\_driver;

`uvm\_component\_utils(my\_driver)

virtual task run\_phase(uvm\_phase phase);

my\_sequence\_item item;

// Peek at the next item without removing it from the queue

if (peek(item) == UVM\_IS\_OK) begin

// Inspect the item or check for certain conditions

// but don't consume it yet

end

endtask

endclass

* 1. get

The get() method is used to fetch and remove the next sequence item from the sequence. This method blocks the driver until an item is available.

Syntax: virtual function uvm\_status\_e get(output uvm\_sequence\_item item);

Example:

class my\_driver extends uvm\_driver;

`uvm\_component\_utils(my\_driver)

virtual task run\_phase(uvm\_phase phase);

my\_sequence\_item item;

// Get the next item from the sequence

get(item); // Block until an item is available

// Process the item

// ...

endtask

endclass

* 1. put

The put() method is used by the driver to send a sequence item back to the sequence. This method is typically called when the driver wants to return an item that was previously fetched or to signal the completion of a sequence item.

Syntax: virtual function void put(input uvm\_sequence\_item item);

Example:

class my\_driver extends uvm\_driver;

`uvm\_component\_utils(my\_driver)

virtual task run\_phase(uvm\_phase phase);

my\_sequence\_item item;

// Get an item from the sequence

get(item);

// Process the item

// ...

// Put the item back to the sequence (optional)

put(item); // Return the item to the sequence

endtask

endclass