DAY 84 - 100 DAYS VERIFICATION CHALLENGE

Topic: UVM Sequences & Sequencers

DAY 84 CHALLENGE:

1. Explain the below methods in uvm_sequence_item class:
    i. get_sequence_id
    ii. set_sequencer
    iii. get_sequence
2. Write a simple uvm sequence template and explain each line.
3. What are pre_body and post_body used in sequence?
4. How does a sequence start?
5. What are the types of sequencer? Explain each?
6. What is p_sequencer, and where is it used?
7. How do sequence, driver, and sequencer communicate?
8. What is the virtual sequence? How is it used?
9. Explain the virtual sequencer and its use.
10. Is it necessary to have virtual sequencer for virtual sequence?
11. What is the difference between sequence & sequencer?

Date
Page

Abhyudaya_UVM2024

DAY : 84

Topic : UVM Sequences & Sequencers

Sol(1) Explain the below methods in uvm_sequence_item class:

i.) get_sequence_id :

This method returns the unique (ID) assigned to the sequence item. Think of it like a special number or code that helps identify which sequence the item belongs to.

ii.) set_sequencer :-

- return m_sequencer handle.

iii.) get_Sequence :

- get sequence item from sequencer using "get" method.

Sol(2) write a simple uvm_sequence templates and explain each line.

```
class Base_test_seq extends uvm_sequence #(tx);
                                  // Sequence item(transaction) type.
  uvm_object_utils(Base_test_seq)
  extern function new(string name = "");
  extern task body();             //every Sequence has a body task
endclass
```

∴ body can:

→ generate transactions
→ start other sequences
→ read/write registers

Soln ③ what is pre_body and post_body used in Sequence?

- All sequence code is implemented in the task body().
- Base class includes prebody and post_body.
- Sequences can be combined using available Sequences
- Its an object and has no run phase.
- uvm_pre_body and uvm_post_body macros are used in uvm to define the code that should be executed before & after the body of a task or function. respectively.
- The main advantage of using uvm_pre_body & post_body macros is that they provide a flexible and efficient way to add functionality to task and function in uvm Testbenches. while also making it easier to maintain and understand the code.

Soln ④ How does a Sequence start?

- Sequence are started on a Sequencer using the built-in Sequence start() method or by using the uvm_do macros.
- Every Sequence has a handle to the sequencer that is running that sequence. Tnant handle is called the m_sequencer handle.

Soln ⑤ what are the types of sequencer? Explain each?

① m_sequencer ② Psequencer

- m_sequencer: A generic Sequencer handle available to all uvm sequences by default. this handle is of type uvm_sequencer_base.
- P_sequencer is a user defined Sequencer or a type specific sequencer.
- This handle is not available by default to a uvm Sequence
- Its created when we register a Sequence to a Sequencer

Sol⑤ what is p_sequencer and where is it used?

- p_sequencer stands for "physical sequencer".
- A physical sequencer is a uvm component that is responsible for managing the flow of transactions between a DUT & the testbench.
- The physical sequencer is typically instantiated as part of the interface btw the DUT and the testbench. and it is responsible for generating the appropriate signals on the interface to communicate with the DUT.
- The p_sequence communicate with the driver to send transactions to the DUT & with the monitor to receive transaction from the DUT.
- The p_sequencer class is a built-in uvm class that extends the uvm_sequencer class and provides additional functionality for managing physical interfaces.
- it includes methods for setting & getting the physical interface. as well as for managing the ordering and prioritization of transactions on the interface.

Sol⑥ How do sequencer, driver and sequencer communicate.?

- The sequencer and driver communicate with each other using a bidirectional TLM interface to transfer REQ & RSP sequence items.
- The driver has uvm_seq_item_pull port which is connected with uvm_seq_item_pull export of the associated sequencer.
- Sequencer send the stimulus to driver via sequencer using handshaking methods send request and wait_for_item_done.

Soln (9) Explain the virtual sequencer and its used?

- A virtual sequencer is a sequencer that is used to control and manage the execution of multiple item and sequence sequencers.
- It is typically used to coordinate the generation of stimulus for a complex system that involves multiple interface and protocols.
- A virtual sequencer is responsible for scheduling and coordinating the execution of multiple item and sequence sequencers based on a set of rules or constraints provided by the testbench.

Soln (10) is it necessary to have virtual sequencer for virtual sequence?

- if you have only a single driving agent, you do not need a virtual sequencer. if you have multiple driving agents but no stimulus coordination is required, you do not need a virtual sequencer.

Soln (11) what is the difference btw sequence & sequencer?

- A sequence_item is a transaction that is sent from a sequence to driver.
- The sequencer handles arbitration btw the sequence driver pair, especially when a sequencer or multiple sequencers are sending concurrent sequence_items to the driver.

DAY 86 - 100 DAYS VERIFICATION CHALLENGE

Topic: UVM Agent, config_db

DAY 86 CHALLENGE:

1. Write a uvm agent template and explain each line.
2. What are Active and Passive modes in an agent?
3. What is get_is_active() method in uvm_agent? Why do we need it?
4. What is uvm_config_db? Why do we need it?
5. What is uvm_resource_db?
6. Explain the difference between uvm_config_db & uvm_resource_db.
7. How set_config_* works?
8. What is the difference between set_config_* and uvm_config_db?
9. Can we use set_config and get_config in sequence?

Adityagi-uvm2024

DAY:86

Topic : Uvm_agent , Config_db

Sol) Write a uvm agent template and explain each line.

- Agent Act as intermediates Btw the testbench and the DUT. They encompass drivers, monitors and sequencers and help in organizing and managing the verification process.

```
class m_agent extends uvm_agent;

  driver drv;
  monitor mon;
  Sequencer sqr;
  Coverage cov;

  `uvm_component_utils ( m_agent)

  function new (string name, uvm_component parent);
      super.new(name, parent);
  endfunction

  function void build_phase (uvm_phase phase);
    drv = driver::type_id::create ("drv", this);
    mon = monitor::type_id::create ("mon", this);
    sqr = sequencer::type_id::create ("sqr", this);
    cov = coverage::type_id::create ("cov", this);
  endfunction

  function void connect_phase (uvm_phase phase);
      super.connect_phase (phase);
    drv.seq_item_port.connect (sqr.seq_item_export);
    mon.ap_port.connect (cov.analysis_export);
  endfunction
endclass
```

Soln ② what are Active and passive modes in an agent?

- in uvm active mode & passive modes have specific meaning related to the way a Component responds to events & transaction in the Testbench environment.
- Active mode: In uvm active mode refers to a mode of operation where a Component initiates transactions or events, such as sending request for driving signals.
- An active Component typically has a task or function that actively generates stimulus. such as driver or Sequencer. or Active agents generate stimulus and drive to DUT.
  - An active agent shall consists of all three Component driver, Sequencer & monitor.
- passive mode: Passive agents sample DUT signals but do not drive them.
  - A passive agent consist of only the monitor or Scoreboard.

Soln ③ what is get_is_active() method in uvm_agent? why do we need it?

- The get_is_active() function is used to find out the type of agent.
- get_is_active() Returns uvm_active if the agent is acting as an active agent & uvm_passive if the agent acting as a passive agent.
- The driver, Sequencer instance are created if its an active agent and monitor instance can be created by default irrespective of agent type.

Soln ④ what is uvm_config_db ? why do we need it ?

- uvm_config_db is a static class that provides a database for storing and retrieving configuration values.
- This database can be accessed from anywhere in the verification enviornment & can be used to pass configuration information Btw components that are not directly connected.
- The uvm_config_db provides two main function : set() and get()
- The set() function is used to set a configuration value in the database. and takes three arguments : The first argument is the value to be assigned to that field and the third argument is the hierarchical path of the component the own the field.
- The get() function is used to retrive a configuration value from the database & take two arguments : first argument is the name of the configuration field. & the second argument is the hierarchical path of the component that owns the field.
- One of main advantages of using uvm_config_db is that it allows configuration values to be passed between components that are not directly connected. for example, a top-level testbench component can set a configuration value using uvm_config_db::set(), and this value can be retrived by a lower-level component that does not have a direct connection to the top-level testbench.

Soln ⑤ what is uvm_resource_db ?

- Its a mechanism used in hardware verification to store & retrive configuration & other data in a standardized way.
- uvm_resource_db is that it allows different components or module within your verification enviroement to easy share & access data without having to establish direct connection or dependency Btw them.

Soln ⓐ Explain the difference Btween uvm_config_db & uvm_resource_db.

- All the methods declared in both them are static in nature so it should be declared with the scope resolution operator as uvm_config_db::set() and uvm_resource_db::set().

declaration of each one of them:

→ uvm_config_db:-

⊙ static function void set(uvm_comp Cntxt, string inst_name, string field_name, T value)

⊙ So uvm_config_db is primarily used in places to acess the resource or parameter where hierarch is important.

⊙ uvm_config_dB is used when you want to set/get resource within uvm Component hierarchy

⊙ The number of uvm_config_db are limited mainly 4 such as get, set, exits and wait_modified

uvm_resource_dB;

⊙ static function void set(input string scop, input string name, T value, input uvm_object accessor = null)

⊙ while uvm_resource_db should be used in places to acess the resource in case of non-hierarchical Context.

⊙ uvm_resource_db is used when you want to set/get resouce except uvm Component hierarchy.

⊙ while the method for uvm resource are more like get_type_get_by_name, read_by_name, read_by_type, write_by_name, write_by_type, set_anonymous.

Soln ⓑ How Set_config_* works?

- when Set_config_* method is called, the data is stored w.r.t string in a table. There is also a global Configurable table.
- These function provides a standardized way to set Configura parameter for uvm Components making the environment more flexible and configurable.

Sol(8) what is the difference Between set_config* and uvm_config_db

- uvm_config_db is a global configration database provided by uvm for passing configuration information between different components in different hierarches.
- it allows components at different levels of the hierarch to share and retrive configuration information.
- use set_config* for direct specific configuration setting at the point where the configuration is needed.

Sol(9) Can we use set_config and get_config in sequence?

- yes we can use set_config* and get_config_* methods in a sequence to set and retrive configuration values for the sequence.
- when you call set_config* on a sequence, it updated the configuration database with the new value for the specified configuration parameter.
- This value can be retrived using get_config* method. By using set_config* & get_config* methods.

example:

```
Class my_sequence extends uvm_sequence #(my_tx);
      `uvm_object_utils (my_sequence)

   function void body();

    If (!uvm_config_db #(int) :: get (this, "", "my_param",
                                      my_param_value))
        `uvm_error (get_name(), "failed to get my_param_v
   end function
endclass
```