1. Write a uvm agent template and explain each line.

A UVM Agent is a component that encapsulates the entire interface logic for interacting with a particular interface in the DUT (Design Under Test). It typically consists of a driver, a sequencer, and a monitor, each responsible for different aspects of the verification process.

class my\_agent extends uvm\_agent;

`uvm\_component\_utils(my\_agent)

// Declare driver, sequencer, and monitor

my\_driver driver;

my\_sequencer sequencer;

my\_monitor monitor;

// Constructor

function new(string name = "my\_agent", uvm\_component parent = null);

super.new(name, parent);

endfunction

// Build phase: Create driver, sequencer, and monitor

virtual function void build\_phase(uvm\_phase phase);

super.build\_phase(phase);

// Create the sequencer, driver, and monitor

sequencer = my\_sequencer::type\_id::create("sequencer", this);

driver = my\_driver::type\_id::create("driver", this);

monitor = my\_monitor::type\_id::create("monitor", this);

endfunction

// Connect phase: Connect the sequencer, driver, and monitor

virtual function void connect\_phase(uvm\_phase phase);

super.connect\_phase(phase);

// Connect driver to sequencer

driver.sequencer = sequencer;

endfunction

// Start phase: Start the sequencer and driver

virtual function void start\_phase(uvm\_phase phase);

super.start\_phase(phase);

// Start the sequencer and driver

sequencer.start();

driver.start();

endfunction

endclass

* class my\_agent extends uvm\_agent;:Defines a class my\_agent that extends the uvm\_agent base class, which is the standard agent class in UVM.
* my\_driver driver;: Declares a driver of type my\_driver to drive signals to the DUT.
* my\_sequencer sequencer;: Declares a sequencer of type my\_sequencer to generate sequences of transactions.
* my\_monitor monitor;: Declares a monitor of type my\_monitor to observe DUT signals and collect transaction data.
* function new(string name = "my\_agent", uvm\_component parent = null);: The constructor of the agent, which initializes the agent with a name and parent component.
* super.new(name, parent);: Calls the constructor of the base class (uvm\_agent) to initialize the agent.
* virtual function void build\_phase(uvm\_phase phase);: The build\_phase is responsible for creating the driver, sequencer, and monitor instances. This phase runs once during the testbench setup.
* sequencer = my\_sequencer::type\_id::create("sequencer", this);: Creates an instance of my\_sequencer (using the factory method type\_id::create), associating it with the agent as the parent.
* driver = my\_driver::type\_id::create("driver", this);: Creates an instance of my\_driver and associates it with the agent.
* monitor = my\_monitor::type\_id::create("monitor", this);: Creates an instance of my\_monitor and associates it with the agent.
* virtual function void connect\_phase(uvm\_phase phase);: The connect\_phase connects the driver to the sequencer, ensuring that the driver gets its input from the sequencer.
* driver.sequencer = sequencer;: Connects the driver to the sequencer by assigning the sequencer to the driver’s sequencer port.
* virtual function void start\_phase(uvm\_phase phase);: The start\_phase starts the sequencer and driver, signaling that the agent is now ready to begin executing transactions.
* sequencer.start();: Starts the sequencer to begin generating transactions.
* driver.start();: Starts the driver to begin processing transactions.

1. What are Active and Passive modes in an agent?

* Active Mode: In active mode, the agent is responsible for generating transactions (e.g., using a sequencer to send transactions to a driver). It actively participates in the stimulus generation process.
* Passive mode: In passive mode, the agent does not generate transactions or apply any stimulus to the DUT. Instead, it only observes signals from the DUT using the monitor and collects data without influencing the simulation's flow.

1. What is the get\_is\_active() method in uvm\_agent? Why do we need it?

The get\_is\_active() method is used to check whether the agent is in an active state or not. It: It helps to determine whether the agent should actively participate in generating transactions. In passive mode, this method would return false, while in active mode, it would return true.

function bit get\_is\_active();

return (this.get\_active() == UVM\_ACTIVE);

endfunction

1. What is uvm\_resouce\_db?

The uvm\_resource\_db is a global database used to store and retrieve resources (configuration settings, data, etc.) across different components in the UVM testbench. It is typically used to share data between components without requiring direct connections, allowing for loose coupling between components.

uvm\_resource\_db#(uvm\_object\_wrapper)::set("resource\_name", my\_object);

1. Explain the difference between uvm\_config\_db & uvm\_resouce\_db.

uvm\_config\_db

* Purpose: Stores configuration settings and parameters for components, typically at the component level.
* Usage: It is commonly used to configure a component before its execution, typically for setting parameters such as timeouts, delays, or other testbench configurations.
* Scope: Configuration is typically passed down from the parent to the child components in the hierarchy.

uvm\_resouce\_db

* Purpose: Stores resources that may be needed globally in the testbench (like common data, state, or references to objects).
* Usage: More flexible in storing any type of object or resource and can be used across components without parent-child relationships.
* Scope: Resources stored in the uvm\_resource\_db are accessible globally to any component in the testbench.

1. How set\_config\_\* works?

* The set\_config\_\* family of methods (e.g., set\_config\_int, set\_config\_string) is used to set configuration parameters at a specific level in the testbench.
* These methods are typically used to set values that will be available to components at or below the specified level in the component hierarchy.

Example:

set\_config\_int("my\_component", "timeout", 100);

1. What is the difference between set\_config\_\* and uvm\_config\_db?

set\_config\_\*

* Used to set configuration parameters that are specific to a component in the testbench.
* It is part of the component’s own configuration system.

uvm\_config\_db

* A more flexible and powerful mechanism for passing data between components, including configuration parameters.
* Can be used for passing parameters up and down the component hierarchy (e.g., from parent to child).

1. Can we use set\_config and get\_config in sequence?

Yes, it is possible to use set\_config to set a configuration value and then retrieve it with get\_config within the same sequence, as long as the configuration is set before calling get\_config in the hierarchy.

Example:

set\_config\_int("my\_agent", "timeout", 100);

int timeout\_value = get\_config\_int("my\_agent", "timeout");