UVM INTERVIEW QUESTION

2025

Prepared by

KITTU K PATEL

+91 9016531681
infoex31@gmail.com

UVM Interview & Application FAQ Handbook

50+ Most Commonly Asked UVM Questions with Descriptive Answers

Prepared by: Kittu K Patel

October 2, 2025

Contents

Preface

This handbook is designed for students, freshers, and working professionals preparing for **UVM (Universal Verification Methodology)** interviews.

It contains over 50 carefully selected FAQs with descriptive answers, practical examples, and important notes. The goal is to provide a single resource that helps you revise concepts and understand applications before interviews or real-world verification projects.

1. UVM Basics

Q: Why do we need UVM methodology?

A: UVM (Universal Verification Methodology) is a standard way to test digital designs. It helps build reusable and organized testbenches. UVM provides ready-made base classes that we can extend and reuse. It also follows a set of rules so different teams can work together easily and understand each other's code.

Key Point:

UVM ensures **reusability**, **scalability**, and **standardization**.

Q: Difference between create and new()?

A: The `new()` method is a constructor for SystemVerilog classes to create an object instance.

The `create()` method of the wrapper class is used to create objects for `uvm_object` and `uvm_component` classes using the UVM factory mechanism.

Factory Benefit:

The UVM factory allows objects of one type to be overridden with derived types without changing the testbench structure (factory override).

Q: What is severity and verbosity in UVM?

A: Severity and verbosity are used for controlling log messages during simulation.

Severity Levels: - UVM_INFO - UVM_WARNING - UVM_ERROR - UVM_FATAL

Verbosity Levels: - UVM_NONE - UVM_LOW - UVM_MEDIUM - UVM_HIGH - UVM_FULL - UVM_DEBUG

Q: What is UVM factory and its use?

A: The UVM factory is used to create UVM objects and components. It allows overriding one type with a derived type at runtime without changing the structure.

Example:

Replace a base sequence with a derived sequence using factory override. This enables maximum reusability and configurability of testbenches.

Q: Why phases are introduced in UVM? What are all phases?

A: In Verilog, modules are static. In UVM, class objects can be created dynamically, so synchronization is needed. The UVM phasing mechanism ensures that objects/components are created, connected, and executed in order.

UVM Phases: 1. Build Phase 2. Connect Phase 3. End-of-Elaboration Phase 4. Start-of-Simulation Phase 5. Run Phase 6. Extract Phase 7. Check Phase 8. Report Phase 9. Final Phase

Q: Difference between `uvm_component` and `uvm_object`?

A: **uvm_component:** - Created once and lives throughout simulation. - Constructor takes 2 arguments: `name, parent`. - Static in nature.

uvm_object: - Created dynamically and may live for shorter duration. - Constructor takes 1 argument: `name`. - Dynamic in nature.

Q: Difference between copy and clone?

A: - `copy()`: Performs a deep copy of the object into an existing object. - `clone()`: Creates a new object (using `create()`) and then performs copy.
Note:

Clone internally calls `create()` followed by `copy()`.

Q: How to run any test case in UVM?

A: Use the command: `run_test("test_name");`

This triggers the UVM phasing mechanism, starting from build phase of the test → environment → components, then run phase, and finally `$finish`.

Q: What approach does build_phase use and why?

A: Build phase follows a **top-down approach**. The parent component is built first, followed by its child components. This ensures proper instantiation and hierarchy setup.

Q: What all UVM phases take simulation time?

A: The phases consuming time are: - Run Phase - Reset (Pre/Post) - Configure (Pre/Post) - Main (Pre/Post) - Shutdown (Pre/Post)

Q: What are bottom-up and top-down approaches in UVM phases?

A: **Top-Down Approach:** Higher-level components are built/configured before lower-level ones. Example: build_phase, final_phase.

Bottom-Up Approach: Lower-level components are processed first. Example: connect_phase, start-of-simulation phase.

Q: What is an active and passive agent?

A: **Active Agent:** Drives stimulus to DUT. Contains driver, monitor, and sequencer.

Passive Agent: Does not drive stimulus. Contains only monitor. Used for coverage and checking.

Q: Why is Objection used in UVM?

A: `uvm_objection` is a mechanism for synchronizing phase completion across multiple components. It ensures simulation continues until all components drop objections, preventing premature phase termination.

Q: What are the different ways of exiting code execution in UVM?

A: 1. Using Objection mechanism

2. Calling `$finish`

3. Using `uvm_fatal`

Q: What is TLM?

A: Transaction Level Modeling (TLM) provides a communication mechanism between components.

Features:

- Unidirectional, bidirectional, or broadcasting connections - FIFO buffering for transactions - Abstraction from signal-level details - Compatible with SystemC

Q: Why do we use TLM_FIFO?

A: TLM FIFO stores transactions between producer and consumer running at different speeds.

Key Benefits:

- Decouples sender and receiver - Prevents blocking when producer is faster - Provides methods like `put()`, `get()`, `try_put()`, `peek()` - Supports bounded and unbounded sizes

Q: What is run_test()?

A: `run_test()` is a UVM method used to start the testbench execution. It constructs the test, initializes the phases, and executes the simulation flow.

Q: What is the use of UVM Factory?

A: The UVM Factory enables the creation and replacement of objects without modifying source code. It supports **polymorphism** and allows factory overrides, which make testbenches highly reusable and configurable.

Example:

We can replace a driver with an extended driver using factory overrides without touching the environment code.

Q: What is the difference between `create()` and `new()` in UVM?

A: `new()` is a standard SystemVerilog constructor that instantiates an object directly. `create()` uses the factory mechanism, enabling substitution of classes through overrides.

Tip:

Always prefer `create()` for UVM components/objects if reusability is desired.

Q: What are UVM phases?

A: Phases in UVM define the execution order of different simulation steps. They ensure proper build, connection, elaboration, and execution of the testbench. Common phases include: `build`, `connect`, `run`, `extract`, `check`, and `report`.

Q: Explain the difference between build phase and connect phase.

A: **Build Phase:** Used for creating and configuring components using factory.
Connect Phase: Used to connect TLM ports/exports between components. This sep-

aration maintains clarity and modularity in testbench design.

Q: What is the use of objections in UVM?

A: Objections are a mechanism to control the end-of-test. Components raise objections when active, and drop them when tasks are done. Simulation ends only when all objections are dropped.

Q: Difference between `set_config_db()` and `uvm_resource_db`?

A: `set_config_db()` is commonly used for setting configuration values specific to a hierarchy. `uvm_resource_db` is a generic database allowing more global access. Best practice: prefer `set_config_db()` for component configurations.

Q: What is a UVM agent?

A: A UVM agent encapsulates a driver, sequencer, and monitor to interact with a DUT interface. It can be **active** (drives signals + monitors) or **passive** (only monitors). This modularity improves reusability across environments.

Q: Difference between active and passive agent in UVM?

A: **Active Agent:** Contains driver, sequencer, and monitor. Generates and drives stimulus. **Passive Agent:** Contains only a monitor. Used for observation without driving DUT signals.

Q: What is a UVM environment (env)?

A: A UVM environment is a container that groups together multiple agents, scoreboards, and other components. It is the top-level reusable block of the verification environment.

Q: What is a UVM test?

A: A UVM test is the topmost component in the hierarchy, derived from `uvm_test`. It configures the environment, applies overrides, and controls stimulus execution.

Q: What is the use of `uvm_config_db`?

A: `uvm_config_db` provides a mechanism to set and get configuration parameters across components in a UVM testbench. It helps in passing values such as virtual interfaces, timeouts, and environment settings in a hierarchical way.

Q: What is a virtual interface in UVM and why is it needed?

A: A virtual interface is a SystemVerilog construct that allows class-based components (driver, monitor) to access DUT signals. It is needed because classes cannot directly connect to Verilog interfaces or signals.

Q: What is the difference between TLM ports and analysis ports?

A: **TLM Ports:** Used for request-response communication between components (bi-directional).
Analysis Ports: Used for broadcasting data to multiple subscribers (uni-directional, non-blocking).

Example:

Monitors use analysis ports to send transactions to scoreboards and coverage collectors.

Q: What is a UVM sequence?

A: A UVM sequence is a class that generates a stream of sequence items (transactions) and sends them to a sequencer. Sequences are reusable and can be layered for complex traffic generation.

Q: Difference between sequence and sequence item?

A: **Sequence Item:** A transaction object that holds data fields (like address, data, control). **Sequence:** A class that generates and controls a series of sequence items. Together, they model stimulus at a high level.

Q: What is a UVM sequencer?

A: A UVM sequencer controls the flow of sequence items from sequences to the driver. It acts as a mediator, ensuring proper arbitration when multiple sequences are running.

Q: What is the difference between `start_item()` / `finish_item(`

and `seq_item_port`?

A: **`start_item()/finish_item()`:** Used in sequences to create and randomize items before sending.

`seq_item_port`: Used in drivers to get the sequence item from the sequencer. Both are complementary: sequence generates, driver consumes.

Q: What are UVM callbacks?

A: Callbacks provide a mechanism to alter or extend component behavior without modifying base code. They enable customization by registering user-defined functions that get executed at specific events. **Use case:**

Modify stimulus generation or monitoring behavior dynamically.

Q: What is a UVM scoreboard?

A: A scoreboard is a verification component that compares expected and actual DUT outputs. It validates correctness of DUT functionality, often receiving transactions via analysis ports.

Q: What is a predictor in UVM?

A: A predictor models DUT behavior and generates expected outputs for comparison in the scoreboard. It helps detect functional mismatches by comparing predicted transactions with DUT responses.

Q: What are the different types of UVM phases?

A: UVM phases are broadly categorized into:

- **Build Phases:** (`build`, `connect`, `end_of_elaboration`)
- **Run-Time Phases:** (`reset`, `configure`, `main`, `shutdown`)
- **Cleanup Phases:** (`extract`, `check`, `report`, `final`)

They ensure structured simulation flow.

Q: What is the difference between task-based and function-based phases?

A: **Task-based Phases:** Allow time-consuming activities (e.g., `run_phase`).
Function-based Phases: Execute immediately without consuming simulation time (e.g., `build_phase`). This separation ensures proper scheduling of simulation tasks.

Q: What is the use of `uvm_object_utils` and `uvm_component_util`

A: `uvm_object_utils`: Registers non-hierarchical classes like transactions or sequences with the factory.
`uvm_component_utils`: Registers hierarchical classes like drivers, monitors, environments. Both macros enable factory creation and field automation.

Q: What is field automation in UVM?

A: Field automation is the automatic implementation of methods like `copy`, `compare`, `print`, and `pack/unpack`. Using macros like `uvm_field_int`, engineers reduce boilerplate code for transactions.

Q: What is the difference between `set_type_override()` and `set_instance_override()`?

A: **`set_type_override()`:** Replaces all instances of a given type with a new type. **`set_instance_override()`:** Replaces a specific instance in the hierarchy. Overrides enable flexible substitution without changing code.

Q: What is the use of UVM factory overrides?

A: Factory overrides allow dynamic replacement of components or objects during runtime. This promotes reusability, configurability, and rapid testbench modifications without editing source code.

Q: What is the difference between analysis export and analysis implementation?

A: **Analysis Export:** Declares the intent to provide analysis transactions.
Analysis Implementation: Defines the actual function that receives and processes transactions. They work together to enable subscriber communication.

Q: What is a UVM subscriber?

A: A subscriber is a component that receives transactions via an analysis export/implementation. It is typically used for coverage collection or transaction logging.

Q: What is the use of `uvm_do` macros in sequences?

A: `uvm_do` macros simplify sequence item creation, randomization, and execution. They reduce verbosity by combining multiple steps into one line of code. **Note:**

While convenient, modern UVM discourages excessive use in favor of explicit control.

Q: What is the purpose of `uvm_config_db#(virtual interface)::set()`?

A: It is used to pass a virtual interface handle from the testbench top to lower-level components like drivers or monitors. This ensures class-based components can drive or monitor DUT signals properly.

Q: What is the difference between `uvm_analysis_port` and `uvm_tlm_fifo`?

A: **uvm_analysis_port:** Broadcasts transactions to one or more subscribers. It is non-blocking and does not store data. **uvm_tlm_fifo:** Provides a queue-like storage mechanism between producer and consumer. It ensures synchronization between differently paced components. Together, they support different transaction passing requirements.

Q: What is the use of `uvm_env`?

A: `uvm_env` acts as a container for agents, scoreboards, and coverage collectors. It provides hierarchy and encapsulation in a UVM testbench, enabling reusability across multiple test environments.

Q: What is the difference between UVM test and UVM environment?

A: **UVM Environment:** Encapsulates agents, monitors, drivers, and scoreboards. **UVM Test:** Top-level class that instantiates environments, applies configurations, and controls simulation flow. Tests decide which environment or configuration to run.

Q: What is `uvm_report_server`?

A: `uvm_report_server` is the central mechanism for handling messages (INFO, WARNING, ERROR, FATAL). It allows customization of logging, filtering, and redirection of output to files or terminals.

Q: What is the difference between `uvm_error` and `uvm_fatal`?

A: **uvm_error:** Reports an error but allows simulation to continue. **uvm_fatal:** Reports a critical error and terminates the simulation immediately. Severity level determines the impact on simulation flow.

Q: What is the use of `uvm_config_db`?

A: `uvm_config_db` provides a centralized way to set and get configuration data across hierarchical components. It simplifies parameter passing (e.g., virtual interfaces, settings) without tight coupling.

Q: What is the purpose of `uvm_set_super_type`?

A: This macro establishes a parent-child relationship between factory-registered types. It is used when creating user-defined classes that need to extend UVM base classes while preserving factory overrides.

Q: What is `uvm_sequence_item`?

A: `uvm_sequence_item` is the base class for transactions. It represents data packets that sequences send to drivers for DUT stimulus. It supports randomization, constraints, and field automation.

Q: What is the difference between sequence and sequencer?

A: **Sequence:** A class that generates and controls the order of transaction items. **Sequencer:** The component that arbitrates and delivers sequence items to drivers. Sequencer acts as a mediator between sequences and drivers.

Q: What is a virtual sequence?

A: A virtual sequence is a high-level sequence that coordinates multiple sub-sequences across different agents. It is typically executed on a virtual sequencer, enabling systemlevel stimulus generation across multiple interfaces.

Conclusion

The objective of this document was to provide a comprehensive collection of Frequently Asked Questions (FAQs) on the **Universal Verification Methodology (UVM)**. With over 50 detailed questions and descriptive answers, this document serves as a practical reference for students, professionals, and verification engineers aiming to strengthen their UVM concepts.

UVM is a powerful methodology that continues to evolve, and having such a consolidated reference helps engineers in interviews, daily work, and building robust testbenches. I hope this document will guide you in both learning and applying UVM effectively in real-world projects.

If you have any queries or need further assistance, please feel free to reach out. I will be glad to help and share my experience with you.

Author: Kittu K Patel
Email: `infoex31@gmail.com`
Note: If you need any help related to UVM, SystemVerilog, or Verification career guidance, feel free to contact me.

— *End of Document* —