UVM Advanced

Course Description

This 4-days course designed for advanced ASIC & FPGA verification engineers that would like to enhance their UVM skills to verify complex digital designs more efficiently.

The training is loaded with extensive practical hands-on labs to verify that the theory is understood plus introduce more use cases than covered in theory slides.

The first day teaches how to create a complex stimuli generation with various advanced techniques. Master-slave protocol is covered in details.

The second day starts with introduction of the advanced synchronization using the callback class, along with barrier synchronization, and the uvm_event class.

The day continuous by covering the end-of-test mechanism, how to raise and drop objections, how to debug UVM objections and how to set TB drain time.

The third day covers the connection reusability between DUT and top TB including BFM, port connection, how to extract RTL parameters vs using package, the bind construct, UVM harness, and races between TB and DUT.

The fourth day covers design patterns, how to create them, what is a singleton pattern.

The training ends by covering how to handle exceptional situation such as reset, error injection and the uvm_heartbeat.

Course Duration

4 days

Goals

1. Generate complex stimuli using master-slave protocol, streaming operator to pack/unpack transactions, matching and different responses, packing dynamic data
2. Synchronize your transactions using the callback class and macros
3. Control the end-of-test mechanism
4. Reuse TB-DUT connectivity
5. Use design patterns
6. Handle exceptional situations

Intended Users

Hardware/Software verification engineers who would like to enhance their skills for ASIC/FPGA designs with advanced UVM techniques.

Prerequisites

1. UVM fundamentals
2. SystemVerilog language
3. Verification guidelines
4. Experience with simulator

Course Material

1. Course book
2. Lab handbook (Phyton notebooks)
3. Virtual Machine with all necessary tools
4. Trainer solutions to all labs

Table of Contents

Day #1

- **Lab #2: Layered Agents**

  - Use the methods and macros of the uvm_packer class to convert data into a bitstream
  - Customize the behavior of the `uvm_packer` class - using metadata
  - Create a translating sequence that converts a higher-level data unit into a lower-level data unit
  - Create components that perform reverse translation
  - Build a layered protocol agent

Day #2

❖ Advanced Synchronization

- Defining a Callback Class
- Implementing Specific Callbacks
- Inserting Callbacks into Component
- Registering Callbacks
- The uvm_event_callback Class
- Barrier Synchronization
- The uvm_barrier Class
- The uvm_barrier_pool Class
- Waiting For an Event – Races
- Persistent Trigger vs. the @ Operator
- The uvm_event Class
- The uvm_event_pool Class

- **Lab #3: Advanced Synchronization**

  - Define custom callback classes
  - Create & register custom callback objects
  - Create named events and trigger them
  - Use uvm_event and uvm_event_callback objects

❖ End-of-Test Mechanism

- Objection Mechanism
- Raising and Dropping Objections
- Propagating Objections
- The set_propagate_mode() Method
- Automatic Raising and Dropping Objections in UVM-1.2
- Debugging UVM Objection
- Drain Time and Timeout
- The phase_ready_to_end() Method

- **Lab #4: UVM Objections**
  - Replace manual raising and dropping objections in sequences
  - Debug issues with objections
  - Improve simulation performance by changing the objection propagation mode
  - Set the drain time and timeout in the test
  - Check if the component is ready to move to the next phase

Day #3

❖ Reusability

- Code reuse
- Horizontal reuse
- Vertical reuse
- D from SOLID - Dependency Inversion Principle
- Passive Environments
- Stimulus Reuse
- The uvm_sequence_library Class
- Configurations Encapsulation
- Interfaces Encapsulation
- Namespace Collisions

- **Lab 5: SoC -Level Testbench**
  - Instantiate the environment at the block-level within the chip-level environment

- Ensure hierarchical configuration
- Create a custom implementation of the uvm_report_catcher class

❖ DUT-TB Connection

- Bus Functional Model (BFM)
- Extracting RTL Parameters
- Tuning TB with RTL Parameters
- Traditional DUT to TB Connection
- SystemVerilog bind Construct
- UVM Harness
- Ports Width Mismatch
- Races between TB and DUT
- SystemVerilog Clocking Blocks
- Clocking Blocks in UVM Environment

- **Lab 6: UVM Harness**
  - Use clocking blocks to mitigate races between DUT and TB
  - Apply the polymorphic interface technique
  - Encapsulate module interfaces into a single reusable interface - UVM Harness
  - Connect the UVM Harness to a RTL module using the “bind” directive
  - Retrieve RTL parameters and pass them to the environment

Day #4

❖ Design Patterns

- Design patterns introduction
- Design Patterns in SW
- Creational Patterns
- Singleton Pattern
- The uvm_coreservice_t Class
- The uvm_root Class
- The uvm_config_db Class
- The uvm_cmdline_processor Class

- **Lab 7: Singleton Pattern in UVM**
  - Access selected UVM services using the uvm_coreservice_t class
  - Modify the environment configuration using the command-line processor
  - Pass data using uvm_config_db in various scenarios
  - Use factory methods to override components in various scenarios

❖ Exceptional Situations

- Reset handling strategies
- Error injection
- Is my TB still alive? The uvm_heartbeat class

- **Lab 8: Handling Exceptions**