Prepared by Mitesh Khadgi

Email: mitesh.khadgi2026@gmail.com

Normal sequence, Proactive sequence and Reactive sequence in UVM

In **UVM (Universal Verification Methodology)**, **sequences** are key components used to model stimulus generation in a testbench. They are executed by **sequencers**, which then communicate with **drivers** to send transactions to the DUT.

There are **three types of sequences** based on how they interact with the DUT and the testbench components:

1. Normal Sequence

- **Definition**: A sequence that generates stimulus in a **default, non-interrupt-driven manner**, assuming the DUT is always ready.
- **Nature**: Straightforward, non-blocking. It simply sends transactions without checking for DUT readiness or waiting for any external event.
- **Usage**: Suitable when the DUT can accept inputs continuously.

Example:

```
virtual task body();
  my_transaction tx;
  repeat (10) begin
    tx = my_transaction::type_id::create("tx");
    tx.randomize();
    start_item(tx);
    finish_item(tx);
  end
endtask
```

2. Proactive Sequence

- **Definition**: A **driver-initiated sequence** that **actively polls or waits for conditions** in the DUT before sending transactions.

- **Nature**: The testbench is *proactively controlling* the flow, possibly waiting for the DUT to be ready.
- **Usage**: Useful when DUT cannot always accept new inputs, and the testbench must control pacing.

Behavior:

- Checks for ready signal or handshaking mechanism before initiating transactions.

Example Pseudocode:

```
task body();
  my_transaction tx;
  repeat (10) begin
    // wait until DUT is ready
    wait_for_dut_ready();
    tx = my_transaction::type_id::create("tx");
    tx.randomize();
    start_item(tx);
    finish_item(tx);
  end
endtask
```

3. Reactive Sequence

- **Definition**: A sequence that **responds to events or outputs from the DUT**, such as acknowledgments, interrupts, or data requests.
- **Nature**: Driven by DUT behavior — it reacts rather than initiates.
- **Usage**: Needed when the DUT controls the data flow, like in interrupt-driven systems or protocols with backpressure.

Behavior:

- Waits for a trigger (e.g., DUT requesting data or sending a signal), and then generates or responds with a transaction.

Example Pseudocode:

```
task body();
```

```
  forever begin
    wait (dut_data_request);  // DUT initiates a request
    my_transaction tx = my_transaction::type_id::create("tx");
    tx.randomize();
    start_item(tx);
    finish_item(tx);
  end
endtask
```

Summary Table

Sequence Type	Driver Role	DUT Role	Flow Control Style	When to Use
Normal	Passive initiator	Passive receiver	Uncontrolled	Simple streaming or stress testing
Proactive	Active controller	Passive receiver	Controlled by testbench	DUT has limited ability to accept data
Reactive	Passive listener	Active initiator	Controlled by DUT behavior	DUT initiates transactions or requests

Overall Types of Sequences in UVM

In **UVM (Universal Verification Methodology)**, sequences are an essential part of stimulus generation and control. They run on **sequencers** and interact with **drivers** to apply transactions to the DUT. Based on **functionality and usage**, sequences in UVM can be categorized in several ways.

Types of Sequences in UVM

1. Virtual Sequence

- **Definition**: A sequence that runs on a **virtual sequencer**, coordinating multiple lower-level sequences across different sequencers.
- **Purpose**: Used to synchronize or control multiple agents or interfaces at once.
- **Example Use Case**: Coordinating read and write sequences on different interfaces like AXI and SPI simultaneously.

```
class my_virtual_seq extends uvm_sequence;

 `uvm_object_utils(my_virtual_seq)


 my_seq1 seq1;

 my_seq2 seq2;


 virtual task body();

  seq1.start(p_sequencer.seqr1);

  seq2.start(p_sequencer.seqr2);

 endtask

endclass
```

2. Parameterized Sequence

- **Definition**: A sequence that is templated with a transaction type.
- **Purpose**: Adds flexibility and reusability for different transaction types.

```
class my_param_seq #(type T=base_transaction) extends uvm_sequence #(T);

 T tx;

 virtual task body();
```

```
    tx = T::type_id::create("tx");
    start_item(tx);
    tx.randomize();
    finish_item(tx);
  endtask
endclass
```

3. Default Sequence

- **Definition**: A sequence that is assigned as the **default sequence** for a sequencer.
- **Purpose**: Automatically starts running when the test begins unless overridden.
- **Usage**:

```
sequencer.default_sequence = my_default_seq::type_id::get();
```

4. Virtual-Only Sequence

- **Definition**: A sequence that doesn't generate real transactions but controls execution flow or calls other sequences.
- **Purpose**: Typically used in test layer to orchestrate sub-sequences.

5. Layered Sequence

- **Definition**: A high-level sequence that starts or coordinates **other sequences**, typically across layers of the testbench (e.g., transaction layer, protocol layer).
- **Purpose**: Useful for protocol stacks (e.g., Ethernet over AXI).

6. Reactive Sequence

- **Definition**: A sequence that **responds to DUT activity**, such as interrupts or acknowledgments.
- **See previous answer** for more details.

7. Proactive Sequence

- **Definition**: A sequence that **actively drives** stimulus based on conditions or status from the DUT.
- **See previous answer** for more details.

8. Randomized Sequence

- **Definition**: A sequence where the transaction contents are randomized (using randomize()).
- **Purpose**: For functional coverage, stress testing, or constrained-random verification.

9. Predefined/Built-in Sequences

UVM provides some built-in sequences like:

- uvm_sequence_base
- uvm_simple_sequence
- uvm_random_sequence
  These are useful as base classes or for simple tasks.

Summary Table

Sequence Type	**Purpose**	**Example Use Case**
Virtual Sequence	Coordinate multiple sequencers	AXI + Ethernet combo tests
Parameterized	Flexible with different transaction types	Reuse with multiple agents
Default Sequence	Auto-start sequence on a sequencer	Basic traffic generation
Reactive Sequence	Respond to DUT activity	Interrupt-based data response
Proactive Sequence	Polls/waits before sending data	DUT readiness handling
Layered Sequence	Drives sequences across abstraction layers	Protocol stack testing
Randomized Sequence	Uses randomize() for transaction fields	Coverage-driven or stress testing
Virtual-Only Sequence	No real traffic, coordinates flows	Top-level test control logic