Seven Separate Sequence Styles Speed Stimulus Scenarios

Sequence design patterns for your UVM stimulus coding toolkit

**The Shaped Sequence**

**Intent:** To control a sequence's generation behaviour programmatically.

**Motivation:** Enabling sequence reuse.

**Applicability:** To any sequence, adding fields to control generation so generalising the sequence.

**Implementation:** Add random members for control fields, non-random members for status fields. The random field values are used to constrain the sequence item(s) generated by the sequence.

**The Configurable Sequence**

**Intent:** To control a complex generalised sequence.

**Motivation:** Ease shaping difficulties with many variables.

**Applicability:** To any sequence, particularly related sequences which share shaping variables.

**Implementation:** Create a configuration that is paired with the sequence. The sequence takes it shaping guards from the configuration object that is randomized outside the sequence.

**The Resourced Sequence**

**Intent:** To access testbench resources.

**Motivation:** To generate stimulus based on information present in the UVM testbench hierarchy.

**Applicability:** To register based sequences and sequences making generation decisions based on design or testbench state.

**Implementation:** A base class assigns to resource handles using the m_sequencer to access the uvm_config_db.

**The Virtual Sequence**

**Intent:** To generate stimulus on several DUT interfaces.

**Motivation:** Most testbenches need to generate and co-ordinate stimulus on two or more interfaces.

**Applicability:** Used to control the stimulus generation in most UVM test cases.

**Implementation:** An extension of the resourced sequence pattern with handles for the target sequences. An init_vseq() method in the test assigns the sequencer handles to the virtual sequence.

**The Hierarchical Sequence**

**Intent:** To create abstraction layers in stimulus generation.

**Motivation:** Improve productivity by defining complex stimulus at a high level.

**Applicability:** To any stimulus generation activity that can be disassembled into a series of sub-tasks.

**Implementation:** Create a library of sequences that implement the task hierarchy with a bottom layer of 'worker' sequences.

'Worker Layer ➔

**The Library Sequence**

**Intent:** To be able to execute one of several sequences at will.

**Motivation:** To support stimulus generation scenarios where random choices of sequences are required.

**Applicability:** Useful when one of several sequences would be valid.

**Implementation:** Create a library sequence that contains an array of sequences extended from a common base type. Select and execute sequences from the array.

**The Layering Sequence**

**Intent:** To transform one abstract stimulus data representation to another.

**Motivation:** It is often more convenient to generate stimulus in one format and then transform it to another.

**Applicability:** Useful any time one type of sequence_item needs to be changed to another. E.g. getting video data to stream on an USB transport layer.

**Implementation:** Insert a sequencer that has a layering sequence running on it. The layering sequence uses the sequencers seq_item_export methods to get the upper layer items, then transforms them before sending to the lower layer sequencer. Multiple layerings can be combined or chained, as illustrated in the example code.