Lab 12: Sequence Stimulus

In this lab, you will use sequences to create stimulus for the counter test bench. You will create the sequence and driver objects, and leverage the ovm_sequencer object from the OVM library. Then you will connect the objects together in the environment.

OVM Sequence Items: `ctr_output` and `ctr_req`

The counter test bench uses two OVM sequence items.

- ctr_output—The counter response transaction. It contains a single data member called `data`: an eight-bit value.
- ctr_req—The counter request transaction. It extends ctr_output and contains an additional data member called `op` which can be set to `inc`, `load`, `nop`, or `reset`.

These ovm_`sequence` items are completely written and are in files `ctr_output.svh` and `ctr_req.svh`.

Create the `test_seq` Sequence

The test_seq sequence creates and randomizes twenty ctr_req objects and passes them to the sequencer. The sequence has been mostly defined in test_seq.svh. Please do the following to complete it:

- Complete the class declaration for test_seq.
- Create a new ctr_req item by calling the constructor.
- Use the start_item and finish_item methods to tell the OVM when you have started creating the sequence item and when it's ready to go into the test bench. These should go around the randomization statement.
- Get the response back from the counter for each request.

While you are responsible for creating the 20 random transactions, the sequence will provide transactions for the reset and will implement a directed test for the counter's rollover behavior.

Create the driver Class

The driver gets sequence items from the `ovm_sequencer` and drives signals on the interface to the counter. Please do the following to complete the driver:

- Create the `driver` class by extending `ovm_driver`.
- Get the next item from the `seq_item_port` at the positive edge of the clock.
- Tell the OVM that you successfully got an item.
- Set the response ID to be the same as the request ID.
- Put the response into the `seq_item_port`.

Finish the counter_env to use the sequence, sequencer, and driver.

The environment puts all the pieces together to connect the sequence to the DUT. Please do the following to complete the test bench:

- Create a declaration for the driver
- Create a declaration for the sequence
- Create a declaration for the ovm_sequencer
- Create a call to the factory to manufacture a sequence
- Create a call to the factory to manufacture a driver
- Create a new sequencer with a call to the constructor.
- Connect the seq_item_port from the driver to the ovm_sequencer
- Start the test sequence.

Run the Test

The test should look like this.

```
# ----------------------------------------------------------------
# OVM-2.0.3
# (C) 2007-2009 Mentor Graphics Corporation
# (C) 2007-2009 Cadence Design Systems, Inc.
# ----------------------------------------------------------------
# OVM_INFO @ 0: reporter [RNTST] Running test verbose_test...
# OVM_INFO @ 20: reporter [test_seq] Sending transaction  op: reset
# OVM_INFO @ 31: reporter [test_seq] Got back: data: 00
# OVM_INFO @ 40: reporter [test_seq] Sending transaction data: 5d op: load
# OVM_INFO @ 51: reporter [test_seq] Got back: data: 5d
# OVM_INFO @ 60: reporter [test_seq] Sending transaction  op: nop
# OVM_ERROR @ 71: ovm_test_top.env.comp [run] FAILED: Expected: data: 5d   Actual: data: 5e
# OVM_INFO @ 71: reporter [test_seq] Got back: data: 5e
# OVM_INFO @ 80: reporter [test_seq] Sending transaction  op: inc
# OVM_ERROR @ 91: ovm_test_top.env.comp [run] FAILED: Expected: data: 5e   Actual: data: 5f
# OVM_INFO @ 91: reporter [test_seq] Got back: data: 5f
```