Digital Design Verification

Lab Manual # 41 – Multichannel Sequences

Release: 1.0

Date: 14-Aug-2024

NUST Chip Design Centre (NCDC), Islamabad, Pakistan

Revision History

Revision Number	Revision Date	Revision By	Nature of Revision	Approved By
1.0	14/08/2024	Amber Khan	Complete Manual	-

Contents

Objective

The objective of this lab is

- To build and connect multichannel sequences to your testbench.
- To build a scoreboard using TLM imp connectors.
- To create a module UVC for the router using the scoreboard.

Tools

- SystemVerilog
- Cadence Xcelium

Instructions for Lab Tasks

The submission must follow the hierarchy below, with the folder named after the student (no spaces), and the file names exactly as listed below.

```
./student_name_lab13/
             ├── task1_mcseq/
             │    ├── tb/
             ├── task2_scb1/
             │    ├── sv/
             │    ├── tb/
             ├── task3_scb2/
             │    ├── sv/
             │    ├── tb/
             ├── router_rtl
             ├── yapp
             ├── hbus
             ├── channel
             ├── clk_and_reset
```

Task 1: Multichannel Sequences

For this task, you will build and connect a multichannel sequencer for the router, and create multichannel sequences to coordinate the activity of the three router UVCs.

Setting Up the Directory Structure

1. First – copy your files from `lab12`/ into `lab13/`. e.g., from the directory in which lab12 and lab13 are placed, type: `cp -R lab12/* lab13/`
   Rename the `lab13/task1_integ` directory to `task1_mcseq`.
   Work in the `task1_mcseq/tb` directory.
2. Create the multichannel sequencer component router_mcsequencer.sv.
   a. Add a component macro and constructor.
   b. Add the references for the HBUS and YAPP UVC sequencer classes.
   The Channel UVCs continuously execute a single response sequence, so they do not need to be controlled by the multichannel sequencer.
   The Clock and Reset UVC could be controlled by the multichannel sequencer if, for example, we wanted to initiate reset during packet transmission. However, for simplicity we'll leave Clock and Reset out of the multichannel sequencer.
3. Create a multichannel sequence library file, router_mcseqs_lib.sv and define a single multichannel sequence, router_simple_mcseq, as follows:

a. Add an object macro and constructor.
b. Add a `uvm_declare_p_sequencer macro to access the multichannel sequencer references.
c. Using the sequences defined in the YAPP and HBUS UVC sequence libraries, create a multichannel sequence to:
    - Raise an objection on starting_phase.
    - Set the router to accept small packets (payload length < 21) and enable it.
    - Read the router MAXPKTSIZE register to make sure it has been correctly set.
    - Send six consecutive YAPP packets to addresses 0, 1, 2 using yapp_012_seq.
    - Set the router to accept large packets (payload length < 64).
    - Read the router MAXPKTSIZE register to make sure it has been correctly set.
    - Send a random sequence of six YAPP packets.
    - Drop the objection on starting_phase.

   There are pre-defined sequences in the UVC libraries for all the above operations.

4. Modify the router_tb.sv testbench to instantiate, build, and connect the multichannel sequencer.

   Hint: Examine a topology report to find the reference for the HBUS sequencer. Remember the connections for the multichannel sequencer references are hierarchical pathnames, not configuration instance name strings, therefore you cannot use wildcard characters in the connection.
5. Create a new test in router_test_lib.sv to achieve the following:
   a. Set a type override for short packets only.
   b. Set the default sequence of all output channel sequencers to channel_rx_resp_seq (copy from a previous test).
   c. Set the default sequence of the Clock and Reset sequencer to clk10_rst5_seq (copy from a previous test).
   d. Set the default sequence of the multichannel sequencer to the router_simple_mcseq sequence declared above.
   e. Do not set a default sequence for the YAPP or HBUS sequencer. Control is now solely from the multichannel sequencer.
6. Add include statements to your top module to reference the new files.

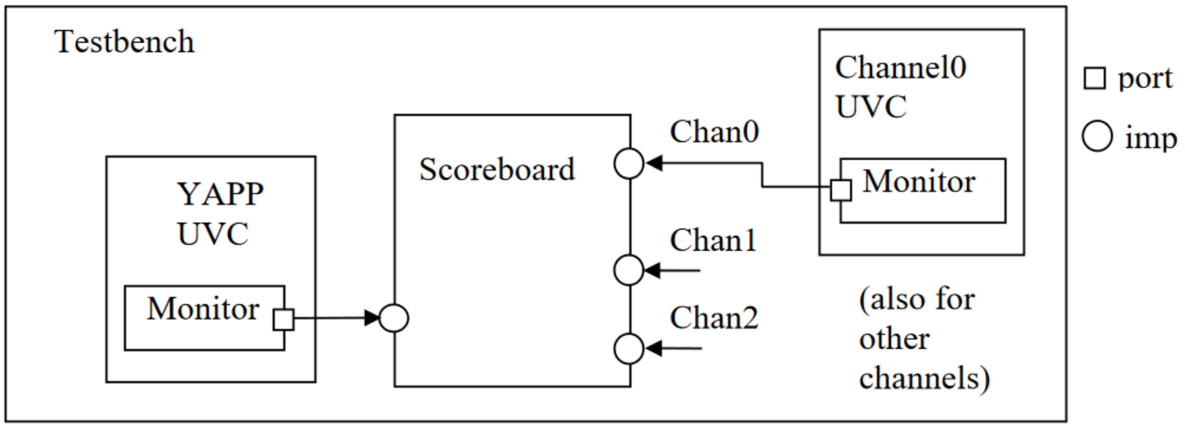
   Make sure the includes are in the correct order, for example the multichannel sequencer must be included before the testbench file, as the testbench creates an instance of the multichannel sequencer.
7. Run a test and check your results. If you open the simulator log file in an editor, you should be able to track packets through the router and see the HBUS read and write transactions. If necessary, you can insert extra delays between the YAPP and HBUS sequences in the multichannel sequence to clearly separate transactions on the different interfaces.

Task 2: Creating a Scoreboard using TLM

For this task, you will build and connect a scoreboard for the router and create TLM analysis port connections to hook up the scoreboard to the UVCs.

The router Module UVC is a complex design, so this lab has been deliberately broken down into separate steps to build the UVC progressively.

The first step is to implement the scoreboard component itself and connect it up to the YAPP and Channel UVCs. For this part of the lab we assume all packets are sent to legal addresses with legal payload length, i.e., the router does not drop any packets.



1. First – copy your files from `lab13/task1_mcseq/tb` into `lab13/task2_scb1/tb`. Work in the `task2_scb1/tb` directory.
2. A TLM analysis port has already been implemented in the Channel UVC monitor for collected YAPP packets. This port is named item_collected_port. Check the Channel monitor in the file channel/sv/channel_rx_monitor.sv to make sure you understand how this port is used.
3. Modify yapp/sv/yapp_tx_monitor.sv to create an analysis port instance.
   a. Declare an analysis port object, parameterized to the correct type.
   b. Construct the analysis port in the monitor constructor.
   c. Call the port write() at the appropriate point
4. In the lab09_sba/sv directory, create the scoreboard, router_scoreboard.sv.
   a. Extend from uvm_scoreboard and add a component utility macro and a constructor.
   b. As the YAPP and Channel UVCs use different packet types, you will need a custom comparison function to compare yapp_packet and channel_packet packets.

      You can use either simple Verilog comparison operators or the uvm_comparer class (see slides and reference material for details).

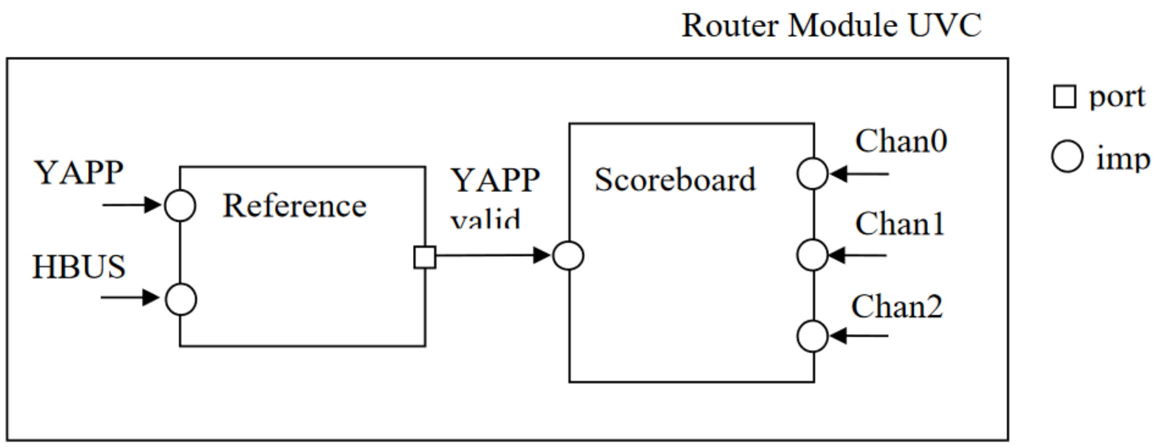
      A simple comparison operator is provided to you in the file packet_compare.sv, copy this into your scoreboard.
   c. Define four analysis imp objects (for the YAPP and three Channels) using `uvm_analysis_imp_decl macros and uvm_analysis_imp_* objects.
   d. Create analysis imp instances in the scoreboard constructor.
   e. Use the YAPP write()implementation to clone the packet and then push the packet to a queue. Hint: Use a queue for each address.

f. Use Channel write()implementations to pop packets from the appropriate queue and compare them to the channel packets, using your custom comparer function.
g. Add counters for the number of packets received, wrong packets (compare failed) and matched packets (compare passed).
h. Add a report_phase() method to print the number of packets received, wrong packets, matched packets and number of packets left in the queues at the end of simulation.

5. In the tb directory, update the tb_top module to include the router scoreboard.
6. In the tb directory, modify the router_tb.sv as follows:
   a. Declare and build the scoreboard.
   b. Make the TLM connections between YAPP, Channel, and scoreboard.
   c. Use a test which generates a good number of legal YAPP packets, i.e. short packets with legal addresses, so that no packets are dropped by the router. We could use the Lab 8 multichannel sequence test or if you did not complete that lab, you could modify the Lab 5 exhaustive sequence test to add the Channel and Clock and Reset sequences.
   d. Check that the simulation results are correct, and debug as required.
7. Once you are happy that the scoreboard is working, check that it correctly reports mismatched packets. You can achieve this by commenting out the short YAPP packet type override in your test class which will allow the YAPP UVC to send oversized packets which will be dropped by the router and create mismatches in the scoreboard. Make sure that your HBUS sequencer is executing hbus_small_packet_seq to set the MAXPKTSIZE register to 20.
   Run a simulation with the type override removed and check the log file for the following:
   - Your YAPP UVC is generating packets of type yapp_packet.
   - The RTL router code generates ROUTER DROPS PACKET messages.
   - Your custom comparer function generates uvm_error messages when the comparison fails.
   - Packets are left in the scoreboard queues at the end of simulation.
   - Your scoreboard reports the number of received, mismatched and matched packets, as well as the number of packets left in the queues. Check the number of packets add up!
8. Write your own custom comparer function which uses uvm_comparer methods instead of Verilog comparisons. Test your new implementation in simulation.

Task 3: Router Module UVC

In reality, the scoreboard will only be one part of a larger router module UVC. For example, the router UVC may also contain reference models and coverage. All these components will be enclosed in an env class.
In our example, we need to know the maxpktsize and router_en register field settings so we know which packets are dropped. We can implement this in a separate router reference model component.
The router reference model connects to the YAPP and HBUS UVC analysis ports and selectively passes on YAPP input packets to the scoreboard depending on the register settings. An env wrapper will instantiate both reference and scoreboard into a single router module UVC, as shown.



1. First – copy your files from `lab13/task2_scb1` into `lab13/task3_scb2`. Work in the `task3_scb2` directory.
2. A TLM analysis port has already been implemented in the HBUS UVC monitor. Note that the HBUS UVC has a common monitor, hbus_monitor.sv, for both the master and slave agents. The HBUS monitor analysis port for collected hbus_transaction's is named item_collected_port.
   Check this to make sure you understand how it is written.
3. Create the router reference, router_reference.sv, in the sv directory.
   a. Extend from uvm_component.
   b. Define two analysis imp objects for the YAPP and HBUS monitor analysis ports, using `uvm_analysis_imp_decl macros and uvm_analysis_imp_* objects. (Copy declarations from your scoreboard.) These are for input data to the reference.
   c. Define one analysis port object for the valid YAPP packets. This is for output data to the scoreboard.
   d. Define variables to mirror the maxpktsize and router_en register fields of the router and update these in the HBUS write() implementation.
   e. In your YAPP write() implementation, forward the YAPP packets onto the scoreboard only if the packet is valid (router enabled; maxpktsize not exceeded; address valid). Keep a separate count of invalid packets dropped due to size, enable and address violations.
4. Create the router module environment, router_module_env.sv, in the sv directory.
   a. Declare and build the scoreboard and router reference components.
   b. Connect the "valid YAPP" analysis port of the reference model to the YAPP analysis imp of the scoreboard model.

5. In the tb directory, modify the router_tb.sv.
   a. Replace the scoreboard declaration and build with the router module.
   b. Modify the TLM connections for the YAPP and Channel analysis ports to allow for the router_env layer.
   c. Add a connection for the HBUS analysis port.
6. Create a router_module.sv package which includes the router module environment, reference and scoreboard files. Import this package into your UVM top module.
   a. Use the same multichannel sequences to test your scoreboard and system monitor implementation.
   b. Check the simulation results are correct and the scoreboard and monitor report the right number of packets.
7. The router module can now be used as a standalone UVC. Copy your router module UVC files to the router directory.