CHIPXPRT

Lab 11

Integrating Multiple UVCs

Module ID : CX-301

Design Verification

Instructor : Dr. Abid Rafique

Version 1.1

Release: 1.0

Document History

The changes and versions of the document are outlined below:

Version	State / Changes	Date	Author
1.0	Initial Draft	Jan, 2024	Qamar Moavia
1.1	Modified with new exercises	March, 2025	Qamar Moavia

Objectives

By the end of this lab you will be able to

- To connect and configure the HBUS UVC, Clock and Reset UVC and three output Channel UVCs.

Tools

- SystemVerilog
- Synopsys VCS

Instructions for Lab Tasks

The required files for this lab can be found on the

```
shared_folder/CX-301-DesignVerification/Labs/Lab11
```

The submission must follow the hierarchy below, with the folder named and the file names exactly as listed below.

```
./Lab11
                ├── task1_integ/
                │   ├── tb/
                ├── router_rtl
                ├── yapp (your own yapp uvc)
                ├── hbus
                ├── channel
                ├── clk_and_reset
```

Task 1: Integrating Multiple UVCs

For this lab, you will connect the `HBUS`, `Clock` and `Reset` and `Channel` UVCs to the `router` DUT.
All three UVCs are provided. None of the UVCs use configuration objects.
These are the directories we will be using for this and subsequent labs:

`hbus/sv`	HBUS UVC files
`channel/sv`	Channel UVC files
`clock_and_reset/sv`	Clock and Reset UVC files
`yapp/sv`	YAPP input UVC (your files from Lab10f)
`router_rtl`	Router DUT

Your current working directory for this lab will be this one:

`task1_integ`	Your working directory for this lab

Setting Up the Directory Structure

1. First – your YAPP UVC is now complete enough to stand by itself. Copy your YAPP files from `Lab10/task1_vif/sv` into `Lab11/yapp/sv`.
2. We will still be working on the testbench, testclass, and top files. Copy the files from `Lab10/task1_vif/tb` into `Lab11/task1_integ/tb`.
   Work in the `Lab11/task1_integ/tb` directory.

Testbench: Channel UVC

1. Update your testbench, `router_tb.sv`, to add the Channel UVCs.
   a. Add three handles of the Channel UVC (`channel_env`) and create the instances in the `build_phase()` method using factory calls.
   b. Use a configuration `set` method to set the `channel_id` property of each Channel instance. The Channel instance for address 0 should have a `channel_id` of 0, the Channel instance for address 1 should have a `channel_id` of 1 and the Channel instance for address 2 should have a `channel_id` of 2. For example,

```
uvm_config_int::set(this, "chan0", "channel_id", 0);
```

Testbench: HBUS UVC

1. Update your testbench, `router_tb.sv`, to add the HBUS UVCs.
   a. Add a handle of the HBUS UVC (`hbus_env`) and create the instance in the `build_phase()` method using factory call.
   b. Use configuration set methods to set the `num_masters` property of the HBUS UVC to 1, and the `num_slaves` property to 0. The HBUS UVC has both master and slave agents. For the router testing, we only need the master agent.

Testbench: Clock and Reset UVC

1. Update your testbench, `router_tb.sv`, to add a handle of the Clock and Reset UVC (`clock_and_reset_env`) and create the instance in the `build_phase()` method using a factory call. This UVC requires no configuration.

Hardware Top Module hw_top

1. Update `hw_top_dut.sv` as follows:
   a. Add an interface instantiation for the `Clock` and `Reset`. The interface file can be found in the `clock_and_reset/sv` directory. Map the `clock`, `reset`, `run_clock` and `clock_period` interface ports to the local signals of the same name.
   b. Connect the `clkgen` module instance to the `Clock` and `Reset` interface instance by replacing the `run_clock` and `clock_period` literal port mappings with the local signals of the same name.
   c. Add interface instantiations for the HBUS and all three Channels. The interface files can be found in the `sv` directory of each UVC directory. Map the ports of the interfaces to the local `clock` and `reset` signals of the same name.
   d. As the `reset` will now be generated by the `Clock` and `Reset` UVC, delete the initial block which generates the `reset` waveform.
   e. Update the port mapping of the router instantiation to connect the Channel and HBUS interface signals.

UVM Top Module tb_top

1. Update `tb_top.sv` as follows:
   a. Add imports for the `Channel`, `Clock` and `Reset` and `HBUS` UVC package files. The packages can be found in the `sv` directory of each UVC.
   b. Set the `HBUS`, `Clock` and `Reset` and `Channel` UVC virtual interfaces to the correct interface. (Hint: The UVC header files contain `typedefs` for each interface.)

      Use wildcards in the pathname to update all UVC components with a single statement.

      Use an absolute hierarchical pathname for the value to select the correct interface instance from the `hw_top` module.

Running Base Test

1. For every UVC (`YAPP`, `Clock` and `reset`, `HBUS` and `Channel`) add the following to your `filelist.f`.
   - An `incdir` reference to the UVC `sv` directory (depending upon the location of UVC in reference to your current working directory)
   - UVC package filename.
   - UVC interface filename.
2. Run a simulation with `base_test` only. Check the topology report carefully to make sure all of your UVCs are instantiated and configured correctly. Copy the topology report into a new file for future reference.

Test Library

1. Add a new test class, `simple_test`, in `router_test_lib.sv` as follows (copy from existing tests). Sequencer pathnames can be read from the topology report.
   a. Set the YAPP UVC to create short YAPP packets with a `set_type_override`.
   b. Set the default sequence of the YAPP UVC to `yapp_012_seq`.
   c. Set the default sequence of each Channel UVC to `channel_rx_resp_seq`.

      Hint: you can set all three Channel UVCs with a single statement.
   d. Set the default sequence of the Clock and Reset UVC to `clk10_rst5_seq`.
   e. Do not define a default sequence for the HBUS UVC.

f. Clean up the test library and remove the older tests. Delete or comment out all the other test classes besides `base_test` and `simple_test`.

Running Simple Test

1. Run a simulation in GUI mode using `simple_test` and verify that the packets transmitted at YAPP UVC interface are correctly received at the interfaces of the corresponding channels. For Example, Packet with address 0 should be received on channel 0 interface and vice versa.

Further Integration Testing

1. Write a new YAPP sequence in the `yapp/sv/yapp_tx_seqs.sv` file to generate packets for all four channels (including the illegal address 3). The packets should have incrementing payload sizes from 1 to 22 and a parity distribution of 20% bad parity (88 packets in total).

   Hint: You could create packets using nested loops for address and payload.

2. Create a new test, `test_uvc_integration`, in the `router_test_lib.sv` file to perform the following:
   a. Set the `run_phase` default sequence of the YAPP UVC to the sequence created above.
   b. Set the `run_phase` default sequence of the HBUS UVC to set up the router with register field `maxpktsize = 20` and enable the router (register field `router_en = 1`).

   Hint: There is a sequence defined for this in the HBUS master sequences `hbus_master_seqs.sv`.

   Hint: The hierarchical path name for the HBUS configuration setting can be read from the topology report.

3. Run a simulation and check the results to see that the three channels are properly addressed, that there is an error signal when parity is wrong, and that packets are dropped if they are bigger than `maxpktsize` or have illegal addresses.