Practical Applications

Table of Contents

1 Before Starting

This document guides through the practical application of verifying the P2S module. The project folder also contains cheat sheets for Verilog and SystemVerilog that you can reference at any time.

1.1 Using DVT IDE for development

Open a terminal and issue the following commands:

$>mkdir dvt_ws

$>export PROJ_HOME=/home/$USER/p2s_2018

$>dvt.sh -workspace /home/$USER/dvt_ws

The code will be developed using DVT IDE. In the DVT window Right-Click in the Navigator bar > Import Project and select the path to p2s_2018. You should see the project in the Navigator bar.

1.2 Using messages for debug purposes

In order to ease debug you can use UVM messages. These can be issued by using the following UVM construct:

`uvm_info(get_name(), <message string here>, <VERBOSITY>)

VERBOSITY can take one of the values: [UVM_NONE, UVM_LOW, UVM_MEDIUM, UVM_HIGH, UVM_FULL].

The message string can be a simple string or obtained by using $sformatf(“C-style string pattern ”, arguments)

1.3 Important notice:

For improved readability, all files should have the name of the class they contain with a **.svh** extension.

2 Day 1

2.1 Theory

- Design and Verification – Digital integrated circuits domain overview
  - OOP concepts refresh
  - Verification Concepts
  - “Banana” diagram
  - Verification components
  - Metrics
  - Divider example - discussion

2.2 Verification Plan

After you read the specification (doc/Parallel_to_Serial_Specification.pdf) you must create a verification plan using and excel spreadsheet.

2.3 Clock and Reset

1) Define the **ex_clk_rst_gen** module with 2 outputs:
- **clk** is the generated clock signal
- **rst_n** is the negative logic, synchronous, reset signal (i.e. active low)

2) Define the CLOCK_PERIOD parameter which determines the clock period

3) Implement the clock generation logic

4) Implement an initialization block for the reset&clock signals

5) Implement a task **drive_reset(*byte length*)** that drives the reset signal for ***length*** clock cycles

You can find examples of Verilog syntax in the Verilog quick reference card (under doc/cs/). You can also you the auto-complete feature in DVT IDE by starting to write something and pressing CTRL+Space.

2.4 Testbench module

1) declare **rst_n** and **clock** signals using wire
2) include the **ex_clk_rst_gen.v** file
3) instantiate **ex_clk_rst_gen** and connect it to the **clk** and **rst_n** wires
4) In an initial block you should call the **ex_clk_rst_gen.drive_reset** task

2.5 How to simulate

In order to run simulations you need to use the simulation scripts from the folder ${PROJ_HOME}/sim. These scrips compile the sources and start a simulation.

$> cd ${PROJ_HOME}/sim

$> ./run.sh <testname> <seed>

Where:

- **testname** - is the name of the test class
- **seed** - is an integer number

In order to analyze waveforms you need to manually add/probe each signal that you want to observe to the waveform.

2.6 Instantiate ex_p2s module

Create an instance of the **ex_p2s** module in the testbench module and connect its clock and reset ports.

2.7 Define interfaces

- Define interfaces for the input and output agents:
  - the interfaces should be defined in files **tb/in/ex_in_intf.sv** and **tb/out/ex_out_intf.sv**

- interfaces should have all the signals (with the appropriate names) as given in the specification document, interface names should be “ex_in_intf” and “ex_out_intf”
- signals should be of type logic

3 Day 2

3.1 Theory

- SV concepts
- UVM concepts

3.2 Interfaces – continue if not done on day 1

3.3 Parallel sequence_item

- create a class called “ex_in_cmd” that inherits from uvm_sequence_item
- copy the code provided in your class
- create an enum called “ex_rnw_t” that can have 2 values: ex_write =0 and ex_read =1;
- create the following items:
  - idle_time – int
  - address – 8bits
  - data – 8bits
  - rnw – type ex_rnw_t
  - make all items randomisable
- create the “compare” function that has one argument of type ex_in_cmd and returns 0 or 1 depending if the argument has the same rnw, data and address as the current object

3.4 Parallel Driver:

- write the “drive_command (ex_in_cmd item)” function that drives the received item on the in_vif virtual interface

3.5 Parallel Monitor:

- write the “monitor_bus()” function that monitors the interface and collects one packet of type ex_in_cmd

3.6 Sequence

- Add a constraint that specifies that the number of transactions can be between 1 and 1000.

4 Day 3

4.1 Parallel Monitor: coverage

- write appropriate coverpoints for the “item_cg” covergroup as discussed during the writing of the Verification Plan

4.2 Serial agent

Similarly to the parallel agent, write a complete serial agent for your application

4.2.1 ex_out_pkg

- import the UVM package and include UVM macros
- include your serial interface (before package definition)
- as more files are created for your serial agent include them inside the package

4.2.2 ex_out_serial_obj (sequence_item)

- inherits from “uvm_sequence_item”

- in addition to the fields found in the parallel “ex_in_cmd”, the ex_out_serial_obj should also have “start_pattern” and “crc” fields
- copy functions from “ex_in_cmd” class
- add a function that computes the crc

4.2.3 ex_out_monitor

- inherits from “uvm_monitor”
- the output monitor should have the following fields:
  - “out_vif” - virtual interface of type “ex_out_intf”
  - “my_item” - of type “ex_out_serial_obj”
  - “new_serial_out_e” - of type event
  - collected_item_port – of type uvm_analysis_port for “ex_out_serial_obj
  - “item_cg” - covergroup sampled at the activation of “new_serial_out_e”
- the Output monitor should have the following methods:
  - constructor: new() - creates all objects
  - build_phase: gets the virtual interface using the uvm_config_db mechanism
  - run_phase: waits for a reset and the starts monitoring items. After receiving an item it will trigger the “new_serial_out_e” event and send the collected item via “collected_item_port”
- any additional methods can be added in order to make the monitor function properly or increase it's readability

4.2.4 ex_out_agent

- inherits from “uvm_agent”
- instantiate a “monitor” object of type “ex_out_monitor”
- implement constructor
- implement build_phase

5 Day 4

5.1 Scoreboard: checks and coverage

- complete the “write_collected_item_in” and “write_collected_item_out” functions. They are called whenever an item of type “ex_in_cmd” or “ex_out_serial_obj” is passed to the specific “uvm_analysis_imp” and handle the received objects.
- instantiate additional “ex_in_cmd” and “ex_out_serial_obj” items as necessary
- implement cover mechanism for scoreboard specific coverage

5.2 Full env simulations

- add the “scoreboard” object of type “ex_scoreboard”
- create the scoreboard during “build_phase”
- connect the monitors from both agents to the scoreboard ports during “connect_phase”
- simulate the whole design and report any found bugs

6 Day 5

6.1 Run Tests and analyze failures using waveforms and messages

6.2 Analyze collected metrics

6.3 Tests – add additional test with only “write” commands

6.4 Wrap-up

6.5 Summer course feedback

Well, that’s about it! Now it’s time to provide us with feedback in order to improve the course contents.