Lab Workbook

SS 2024

Industrial Hardware Verification

Jakob Lechner
Markus Ferringer

SystemVerilog / UVM

Contents

1 SystemVerilog / UVM **3**
1.1 Exercise 1: Add Reset Logic to AVMM Driver (1 Point) 3
1.2 Exercise 2: Connect Analysis Ports (1 Point) 3
1.3 Exercise 3: Extend Coverage Declarations (2 Points) 4
1.4 Exercise 4: Complete Password Sequence and Extend Test (2 Points) 4
1.5 Exercise 5: Reach Full Coverage (3 Points) 5
1.6 Exercise 6: Break the UUT (1 Points) 5

Lab Mode

- You have to elaborate these exercises on your own. This lab part is **not** a team effort.
- We provide stubs containing basic templates for you which are to be extended with your implementations.
- Source code segments where code needs to be added are marked with **TODO** comments.
- We strongly recommend to read through an entire task before starting with the implementation (including the questions section). You will sometimes find hints later that might simplify things.
- The testbench project needs commerical QuestaSim licenses. These are available on the workstations of the ECSLAB.
- To compile and run the simulation go to directory *sim* and execute `make`.
- In case you want to inspect the waveform after running the simulation execute `make wave`.
- Check that there are no errors or warnings reported by the testbench. If your modifications cause any errors or warnings, fix them before continuing with the exercises.
- Once done with all exercises, please execute `make clean`, zip **everything** (including makefiles, scripts, and frameworks) and submit it via TUWEL

Chapter 1

SystemVerilog / UVM

For these exercises we will be using a UVM testbench to verify a SytemVerilog module named *avmm_mem*. This module implements a byte memory with a depth of 256 entries. The memory is divided into two pages, spanning the following address ranges:

- Page 1: Address 0 to 127.
- Page 2: Address 128 to 255.

The memory can be accessed through an Avalon-MM interface. Elements of Page 1 can be written and read at any time through valid R/W operations on the AVMM interface. Elements of Page 2, however, are write-protected. To disable write protection a passcode needs to be written to the last two elements of Page 1:

- Set memory value of address 127 to 23.
- Set memory value of address 126 to 79.

1.1 Exercise 1: Add Reset Logic to AVMM Driver (1 Point)

- Open file tb/avmm/avmm_driver.sv
- At the beginning of the *run_phase* task, add statements to drive zero values on all input signals of the AVMM interface associated with that driver. This will ensure that the UUT inputs are properly defined during the reset phase.
- After the reset assignments add code the the *run_phase* task that waits until the reset signal of the UUT gets deasserted.

1.2 Exercise 2: Connect Analysis Ports (1 Point)

- Open file tb/avmm_mem/avmm_mem_env.sv
- Locate the *connect_phase* function and connect the analysis port of the AVMM agent with the analysis imp ports of the coverage class (m_coverage) and the scoreboard (m_scoreboard).

1.3 Exercise 3: Extend Coverage Declarations (2 Points)

- Open file tb/avmm_mem/avmm_mem_coverage.sv
- Locate the covergroup *m_cov* and review the existing coverpoints.
- Use crosses to create bins for the following scenarios:
  - Read access happens for all read values specified in coverpoint *cp_readdata*, with all addresses specified by coverpoint *cp_address*.
  - Write access happens for all write values specified in coverpoint *cp_writedata* with all addresses specified by coverpoint *cp_address*.
- Create transitional coverage with a bin for each of the following scenarios:
  - Read access after read access.
  - Read access after write access.
  - Write access after read access.
  - Write access after write access.
- Specify coverage where a read access occurs after a write access, both accesses using the same address, for all addresses specified by coverpoint *cp_address*. The write access shall happen while the write access to Page 2 is locked.
- Specify coverage where a read access occurs after a write access, both accesses using the same address, for all addresses specified by coverpoint *cp_address*. The write access shall happen while the write access to Page 2 is unlocked.

The coverage class contains some provisions to help with the coverage declarations:

- There is a function named *get_access_type*, which can be called when defining a coverpoint. The function returns whether a read or write access was performed. In case the parameter *check_page_2_locked* is set to 1, the return value of the function specifically returns whether the write access happened while Page 2 was locked or unlocked.
- For the last two coverage declarations it is necessary to check if two successive Avalon accesses where performed with the same address. The coverage class, therefore, not only stores the current transaction (m_item) but also the transaction that occurred before that (m_prev_item). Using these two transaction objects you can compare if the addresses were identical. This could, e.g. be done as a condition in your coverpoint or cross declaration:

  cross_a : **cross** < *list of coverpoints* > **iff** < condition >.

  The condition acts a guard which ensures that bins of the cross are only updated if the condition is satisfied at the sampling point.

1.4 Exercise 4: Complete Password Sequence and Extend Test (2 Points)

- Open file tb/avmm_mem_test/avmm_pw_sequence.sv. This file implements a sequence for unlocking or locking Page 2 write access.

- Complete the code in task *body* to implement the unlock sequence.
- Open file tb/avmm_mem_test/avmm_mem_test.sv
- Extend the task *run_phase* to execute not only the sequence *avmm_default_seq* but also the password sequence *avmm_pw_seq*. Use the *m_set_password* property of the password sequence to control whether the password should be set or cleared.

1.5 Exercise 5: Reach Full Coverage (3 Points)

- Open file tb/avmm_mem/avmm_mem_tr.sv
- Add constraints to help achieve a coverage of 100%.
- Run the simulation and check the generated coverage report (sim/covhtmlreport/index.html).
- In case the coverage still is below 100%, you have two options:
  1. Optimize your constraints.
  2. Increase the number of generated transactions by increasing the value of *m_test_iterations* and/or *random_seq.m_transaction_count* in *avmm_mem_test*.
- Continue tweaking your testbench until you reach a coverage of 100%.

1.6 Exercise 6: Break the UUT (1 Points)

- Open file uut/avmm_mem.sv
- Modify the code somehow to introduce a bug that breaks the function of the module.
- Rerun the simulation and check if the scoreboard reports any errors.
- Comment out the bug you introduced and restore the correct functionality.