Accelerating Systems with Programmable Logic Components – 1DT109

2023 – HT1-HT2

Lab 3 – Verification with SystemVerilog for a Matrix Multiplier

Overview

In this lab you will have the opportunity to write a testbench using the SystemVerilog verification methodology to verify the matrix multiplier that you have designed in Lab 2.

Tools

Xilinx Vivado HLx Edition 2018.3

Intended Learning outcomes

- Understanding the object-oriented methodology for verification.
- Understanding the basics of SystemVerilog techniques including:
  - Class
  - Random variables
    - Rand/Randc. You can use both or just one of them)
    - Constrained randomization
  - Assertions
    - Current assertions
    - Immediate assertions
  - Function/FSM coverage
    - You can choose to report either functional coverage OR FSM coverage.
- Note that you are required to write your testbench to cover ALL of the above points.

Assessment

- Your design needs to produce the expected result using the SystemVerilog methodology covered in the lectures Verification I and Verification II.
- You should demonstrate your solution, and explain your design to the lab assistants before the second week in Period 2 (before week 45).
- In order to not get overloaded, we recommend that you start early, make good use the first two lab sessions, and only leave the demonstration to the last lab session if possible (you can present at an earlier session as well).
- Both students must understand all parts of the solution by themselves (this will be checked), and both students must be present during the demonstration. If you cannot both be present during any of the lab sessions for demonstration

and have a valid reason, get in touch with the teaching assistants to book another time slot.

1 Testbench for Lab2

The current testbench for Lab2 consists of three main part:

1, Matrix initialization

```
for (row = 0; row < 2**MATRIX_DIMENSION_LOG_2; row = row + 1) begin
    for (column = 0; column < 2**MATRIX_DIMENSION_LOG_2; column = column + 1) begin
        matA [(2**MATRIX_DIMENSION_LOG_2)*row + column] = $urandom;
        matB [(2**MATRIX_DIMENSION_LOG_2)*row + column] = $urandom;
        matRSW [(2**MATRIX_DIMENSION_LOG_2)*row + column] = 0;
        matRHW [(2**MATRIX_DIMENSION_LOG_2)*row + column] = 0;
```

2, Calculate reference result matrix

```
for (row = 0; row < 2**MATRIX_DIMENSION_LOG_2; row = row + 1)
    for (column = 0; column < 2**MATRIX_DIMENSION_LOG_2; column = column + 1)
        for (count = 0; count < 2**MATRIX_DIMENSION_LOG_2; count = count + 1)
          matRSW [(2**MATRIX_DIMENSION_LOG_2)*row + column] =
                                matRSW [(2**MATRIX_DIMENSION_LOG_2)*row + column] +
                                matA [(2**MATRIX_DIMENSION_LOG_2)*row + count] *
                                matB [(2**MATRIX_DIMENSION_LOG_2)*count + column];
```

3, Feed initialized matries to the hardware matrix multiplier

```
repeat (2) begin
#20
s00_axis_tvalid = 1;
for (row = 0; row < 2**MATRIX_DIMENSION_LOG_2; row = row + 1) begin
    for (column = 0; column < 2**MATRIX_DIMENSION_LOG_2; column = column + 1) begin
        // Should use non-blocking assignment here
        s00_axis_tdata <= (sel == 0) ?
                                matA [(2**MATRIX_DIMENSION_LOG_2)*row + column] :
                                matB [(2**MATRIX_DIMENSION_LOG_2)*row + column];
        // set the last signal when sending the last data item
        if (row == 2**MATRIX_DIMENSION_LOG_2 - 1 &&
                                column == 2**MATRIX_DIMENSION_LOG_2 - 1)
            s00_axis_tlast = 1;
        #20;
        // wait until the slave is ready to read the data
        while (!s00_axis_tready) begin
            #20;
```

4, Kick starts the accelerator and compare the output matrix from the hardware matrix multiplier with the reference result matrix. Report errors if there are any.

```
// start the accelerator
#20
start = 1;
#20
start = 0;

// wait for the reslt to arrive from the accelerator
m00_axis_tready = 1;
row = 0;
column = 0;

while (!m00_axis_tlast) begin // exit if last data already received
  #20;
  if (m00_axis_tvalid == 1) begin // valid data on the bus
    matRHW [(2**MATRIX_DIMENSION_LOG_2)*row + column] = m00_axis_tdata;
    column = column + 1;
  end
  if (column == 2**MATRIX_DIMENSION_LOG_2) begin
    column = 0;
    row = row + 1;
  end
end

m00_axis_tready = 0;
count = 0;

// compare the hardware and software results
for (row = 0; row < 2**MATRIX_DIMENSION_LOG_2; row = row + 1)
  for (column = 0; column < 2**MATRIX_DIMENSION_LOG_2; column = column + 1)
    if (matRSW [(2**MATRIX_DIMENSION_LOG_2)*row + column] !=
                                    matRHW [(2**MATRIX_DIMENSION_LOG_2)*row + column]
      ||^ matRHW [(2**MATRIX_DIMENSION_LOG_2)*row + column] === 1'bX) begin
      count = count + 1;
      $display ("HW/SW result mismatch! …
```

2 Your tasks

As you may have already observed, the testbench is done via procedure-oriented programming, that is, the program uses for-loop as the only way to generate test matrix and compare the results.

In this lab, please consider change this testbench with the SystemVerilog techniques which enable object-oriented verification. Particularly:

1. Please implement the testbench as a class, such as `class tb`.
2. Please implement the four functionalities listed in Section 1 use class membership functions instead of dangling for-loops which sit outside a class. The goal is that you can call functions in `class tb` to do all the four steps listed in Section 1.
3. During matrix initialization, please use class random variables to generate the random numbers. Please also use constrains on the numbers being generated. But also, be sure to test the matrix multiplier with as random as possible. Design your constrain groups and motivate the rules to the TA.
4. In the testbench, the AXI bus timing property of the matrix multiplier is not verified. In your new `class tb`, please verify that the **output** of the matrix multiplier obeys the timing property of the AXI stream bus. Please use concurrent assertion for this task.
5. Lastly, please report the FSM or functional coverage of your matrix multiplier. Are there any unreachable FSM state(s) or state(s) that goes wrong with the FSM state transition graph? Are there any state(s) that is holding with not enough cycles, etc.