University of Tehran

Electrical and Computer Engineering Department

ECE (8101) 342

Object Oriented Modeling of Electronic Circuits – Spring 1404

Homework 3: SystemC RTL – RTL and BFM
Due Date: Ordibehesht, 15 1404

SystemC RTL design
Incorporating an RTL design into an upper-level design
Formation of SystemC BFM equivalent to RTL

In this homework you will use the SystemC environment, so it is important that you download and install the SystemC library in your C++ environment. Download SystemC from Accellera.org.

Description: RTL Processing Element

1. We start with developing a Radix-2 8-bit base sequential multiplier with start (*startB*) and ready (*readyB*) handshaking signals. The circuit has *inA[7:0]*, *inB[7:0]*, *outW[15:0]* inputs and outputs. Note that other alternatives for this design are an array multiplier, a Radix-4 sequential multiplier and several others. Show RTL schematic diagram of this circuit and develop the SystemC RTL description for it. Test your multiplier in SystemC using a testbench in SystemC.

2. You will now use the base multiplier of Part 1 in an approximate multiplier that takes two 16-bit operands and produces a 16-bit result. As in Part 1, this multiplier also has start and ready signals for its handshaking that we refer to *startA* and *readyA*. The inputs are *opndA[15:0]* and *opndB[15:0]* and the output is *multW[15:0]*. Show the complete schematic of the RTL design, write SystemC RTL description of the approximate multiplier and develop a testbench for it. Refer to the details below for the method of designing this multiplier.

3. Generate a cycle-accurate Bus Functional Model of the multiplier of Part 2. Use the same testbench as in Part 2 for testing it.

4. Exercise various ways this multiplier can be built and see how the BFM model can be adjusted to reflect the implied hardware.

Approximate Multiplier:

For performing this multiplying and only using 8 bits of A and B, a basic yet highly inaccurate method involves extracting the eight most significant bits of *opndA* and *opndB*, multiplying them with the 8-bit multiplier (Part A), and using the 16-bit result.

Alternatively, instead of simply selecting the leftmost eight bits of *opndA* and *opndB*, we can identify the first '1' from the left in the operands and select the eight most valuable bits to its left. This approach disregards several less significant bits on the right-hand side of the two operands. However, if the most valuable bit of an operand is not within the eight most significant bits, the rightmost bits of that operand are retained without omission. We call this multiplier a "valuable 16-bit approximate multiplier" (VAM16).

To implement this multiplier, begin by locating the first '1' from the left in each operand. If this '1' is found within the upper eight bits, select the subsequent eight bits for the multiplication. Keep track of the number of ignored bits for each operand. Proceed by multiplying the two eight-bit operands and then append zeros to the right of the result. The number of appended zeros should equal the total count of ignored bits from both operands.

For additional details, please refer to the following paper: Z. Hojati and Z. Navabi, "A Low-Cost Combinational Approximate Multiplier," 2023 26th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS), Tallinn, Estonia, 2023, pp. 136-139, doi: 10.1109/DDECS57882.2023.10139501.