CST456 – Lab 3

Simple Randomized Testbench with Scoreboard and Checker

Objective

The objective of this lab is to implement a simple randomized testbench to test the operations and operands of a synchronous arithmetic and logic unit (ALU).

Design Under Verification (DUV)

The ALU contains the following ports:

Port Name	Signal Type	Direction	Number of Bits
clk	Clock	Input	1
op_start	Control	Input	1
operation	Control	Input	2
operand_a	Data	Input	8
operand_b	Data	Input	8
result	Data	Output	16

At the rising edge of the clock, if op_start is asserted (== 1'b1) then the operation, operand_a, and operand_b signals are latched into the ALU. Exactly two clock cycles later the result of the operation is read from the result port. The ALU supports the following operations:

Name	Value	Description
ADD	2'b00	operand_a is added to operand_b. Only the lower 9 bits of the result are used.
MULT	2'b01	operand_a is multiplied with operand_b. This is an unsigned operation and all 16 bits of the result are used.
OR	2'b10	operand_a is bitwise ORed with operand_b. Only the lower 8 bits of the result are used.
AND	2'b11	operand_a is bitwised ANDed with operand_b. Only the lower 8 bits of the result are used.

Testbench

Edit the tb.sv file and create a testbench to test all the operations for a randomly generated subset of operand values. The testbench will include a scoreboard struct to log the operation, operand_a, operand_b, and result values, as well as a checker task that checks the scoreboard once an operation has completed.

Create a repeat loop to continually generate random values for the operation, operand_a, and operand_b signals. Use the std::randomize function and the `RND_CHECK macro to check that randomization was successful. At the appropriate times log the operation, operand_a, operand_b, and result values to the scoreboard, and call the checker task once the scoreboard has been completed. Use the `FAIL_UNLESS_EQUAL macro to check the expected result with the tested result. Note that the `FAIL_UNLESS_EQUAL macro does not stop the test in the event of a failure. Also add necessary code to your testbench so that functional coverage and code coverage reports will be created after the run.

To run the simulation in Xilinx Vivado for Windows execute following commands in cmd window under SIM directory:

```
call C:\Xilinx\Vivado\2023.2\bin\xvlog --sv ../SRC/duv.sv ../SRC/tb.sv

call C:\Xilinx\Vivado\2023.2\bin\xelab -debug typical -top tb -snapshot duv_tb_snapshot

call C:\Xilinx\Vivado\2023.2\bin\xsim duv_tb_snapshot -R

call C:\Xilinx\Vivado\2023.2\bin\xsim  duv_tb_snapshot --tclbatch xsim_cfg.tcl

call C:\Xilinx\Vivado\2023.2\bin\xsim --gui  duv_tb_snapshot.wdb

Code Coverage:

call C:\Xilinx\Vivado\2023.2\bin\xelab -svlog ../SRC/duv.sv -svlog ../SRC/tb.sv -cc_type sbct -cc_db DB1 -cc_dir ./cRun1 -R

call C:\Xilinx\Vivado\2023.2\bin\xcrg -cc_db DB1 -cc_dir ./cRun1 -cc_report ./cReport1
```

Lab Submission

Zip the entire contents of the lab directory and submit it to Canvas.