CST456 – Lab 2 – Part 2

Layered Testbench

Objective

The objective of this lab is to employ a SystemVerilog interface to implement a layered testbench that exhaustively tests 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	8

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 upper eight bits of the result are read from the result port. The lower eight bits of the result are read from the result port one clock cycle later. The op_start signal is ignored during the middle of an operation. The ALU supports the following operations:

Name	Value	Description
ADD	2’b00	operand_a is added to operand_b. Only the lower nine bits of the 16-bit result are used.
MULT	2’b01	operand_a is multiplied with operand_b. This is an unsigned operation and all sixteen bits of the result are used.
OR	2’b10	operand_a is bitwise ORed with operand_b. Only the lower eight bits of the 16-bit result are used.
AND	2’b11	operand_a is bitwise ANDed with operand_b. Only the lower eight bits of the 16-bit result are used.

Figure 1 shows a timing diagram for the addition operation, where 255 + 255 = 510.

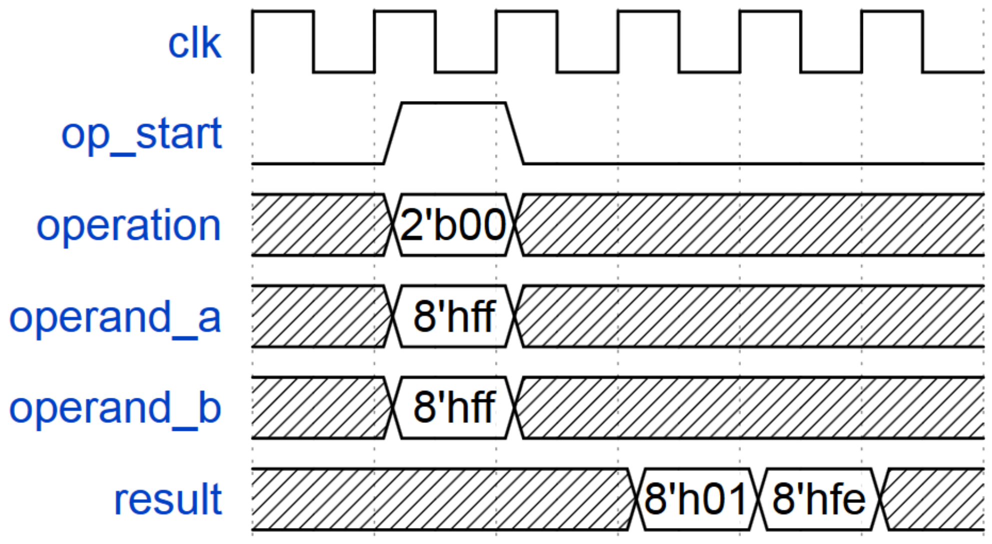

Figure 1 - Timing diagram for an addition operation.

Package

Rather than using the binary value for the ALU's operations, it is more human-readable to use an enumerated type. Edit the typedef_pkg.sv file and create an enumeration typedef for the four ALU operations: ADD, MULT, OR, and AND. Name the typedef operation_t. This typedef will be imported into the interface and testbench.

Interface

We will use an interface to connect the testbench to the DUV and to provide a task that performs ALU operations. Edit the interface.sv file and import the operation_t typedef from the package. Declare the signals, other than clk, that will connect to the DUV and create a task called execute_op with the following ports:

Port Name	Data Type	Direction
op	operation_t	Input
op_a	logic [7:0]	Input
op_b	logic [7:0]	Input
res	logic [15:0]	Output

The task performs the ALU operation specified by op using the two operands op_a and op_b. It returns the result of the operation through the res port.

Testbench

Edit the tb.sv file and create a testbench to exhaustively test all the operations and operand combinations that the ALU supports. Create a do-while loop and two nested for-loops. The do-while loop enumerates through all four operations, the first for-loop enumerates through all 256 values for operand_a, and the second for-loop enumerates through all 256 values for operand_b. Use the interface’s execute_op task to send data to and receive data from the DUV, and use the `FAIL_UNLESS_EQUAL macro to check the expected result with the tested result. The DUV does not contain any bugs, so the testing is expected to be successful.

Simulate your testbench in Xilinx Vivado (**version 2023.2**) using following commands:

(simulation commands must be run under SIM directory)

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

Lab Submission

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