**Minilab 2 Proposal**

ECE554, Fall 2022

In this mini lab, we’re building a MAC that will be used in an 8\*8 systolic matrix multiplier. Because of the way systolic multiplication works, we need to ripple both inputs and the result to the other MAC units. Specifically, the TPU MAC unit performs multiplication and accumulation on the input of two 8-bit data “Ain” and “Bin”. The MAC has a “WrEn” signal which controls the storing of the first data we want to use for first time computation into a 16 bit register with the data input from “Cin”. The computation result will be output to “Cout”, which is 16 bit because we eventually multiply two 8-bit data; the MAC unit also has two 8-bit registers that serves to output “Ain” and “Bin”. Note that all registers are controlled by an enable signal “en”.

We implemented modules such as D flip flop with enable signal (dff\_en.sv), configurable bit width register (with default width 8, named register.sv), and MAC for TPU (tpumac.sv). Our test cases will test these modules’ behaviors with respect to corresponding inputs. For example, we implemented our test cases to test if the output is correct, if the output is updated under “en” signal, and if the “Cin” updates under WrEn signal.

For the test bench to work, we need to initialize the test bench by giving all variables an initial condition such as reset and initial input value. We first established clock signal, which alternates every 5 nanoseconds. Then we gave negative reset signal “rst\_n” an initial state of 1 to avoid having not sure values; we then let “rst\_n” to be 0 at the negative edge of clock to clear all flip flops and the register as a good practice. For the other control signal “WrEn”, we asserted register write signal “WrEn” to store Cin into the register for first time operation, and de-asserted it after the Cin data was stored into the register. Lastly, we gave two inputs “Ain”, “Bin”, and “Cin” initial values 0, and set the counter to number of errors to 0.

After configuring all control signals and initial input values, we start testing the TPUMAC with different test cases. We are having 8 cases in total, each one testing randomly generated number combinations and control signal combinations. The general logic behind these cases is that we first test if any of the registers output incorrect values, and then check if the output “Cout” is computed correctly. Specifically, we check register by initializing all input test values, disabling all registers to avoid writing into them after initialization, and lastly check if the input value is the same as output value from the register. To check the “Cout” value, we compare the output from tpumac module to the value computed in the test bench using Ctest = (Atest \* Btest) + Ctest.

If in any test cases there’s an error, we increment the error count by one so at the end of loop we know there’s something wrong with the tests and we stop the test bench and print “# Error @xxx: Result not computed correctly” or “Error @xxx: Values were not stored in register” with respect to different errors. Printing the detailed error messages out helps us to find the error location which facilitates debugging. If there’s nothing wrong with the code, we print “Mission Successful!: Passed all tests!!!” to celebrate the success.

The result is, of course, we passed all test cases thanks to the random number testing which, again, will find out any computation errors of Cout. For register storing/enable testing, we test register storing capabilities for every for loop cycle and that will also ensure functionality of register enable signal.

Appended is a picture of testbench result.

Graphical user interface

Description automatically generated