Lab 5

An 8-Bit Multiplier in SystemVerilog

In this experiment, you will design a multiplier for two 8-bit 2's compliment numbers and implement it using SystemVerilog on the DE2 FPGA board.

Assignment

- Read the Lab 5 description (**UPDATED: 09/18/17**) in the lab manual and complete the Lab 5 Pre-Lab before the lab section.
- Work on Lab 5 report and Lab 6 Pre-Lab after the lab section.
- View the report outline for guidelines on writing the report.

Demo

- Functional simulation completes successfully. It should include multiplications using different signs (e.g., 7*59, -7*59, 7*-59, -7*-59) (1 point).
- ClearA_LoadB works on the board. (1 point)
- Multiplication works. We will test + * +, + * −, − * +, and − * −. (1 points)
- Consecutive multiplications work, like -1 * -1 * -1 * ... . **ClearA_LoadB or Reset will not be pressed between two presses of Run.** (1 points) (**UPDATED: 09/18/17**)
    - Note - this is only expected to work if the product from the previous multiplication operation can be truncated to fit into 8-bits without changing the value (e.g. the product is within the range [-128,127])
    - -1 (16'b1111111111111111) is truncated to (8'b11111111), which is still interpreted as -1, so this is expected to work for consecutive multplications.
    - 256 (16'b0000000100000000) is truncated to be 8'b00000000, which is not interpreted as 256, so this is not expected to work for consecutive multiplications.
- Execution cycle responds correctly – exactly one execution per press of the "Run" button. (1 point)

Updates

- We are trying to make this course better by fixing errors and updating the documents. However, this results in several differences between the latest online version and the printed lab manual.
- **09/18/17** Lab 5 description: Added more descriptions and changed the demo points.

FAQ

- **"How do we copy over Pin assignments from our last lab?"**
    - The .qsf file in you project stores the pin assignments.

- **"Can you elaborate on number 4 of the required demo listed above?"**
    - It just means that your circuit should run the multiplication several times without resetting the circuit (for example, we'll do 2x2=4 first, then 4x2=8, then 8x2=16 and so on by only pressing the 'Run' button several times without pressing the 'Reset' or 'ClearA_LoadB' in between). Note that you will not need to guarantee correct operation if the result XAB doesn't fit into 8-bits, **bu t negative numbers should still work for consecutive multiplies**.

- **"What's the point of "reset?" In our state machine it cycles back to the initial/start state after completing the full operation, so I don't see why it's useful."**
    - You're right that 'Run' and 'ClearA_LoadB' are pretty much all that you need. But it's good to have a reset capability for any circuit. That's why your computer has the 'restart' option on top of the 'shut down', right?
- **"Suppose we hold down the run button through an entire cycle; it should wait for the run button to go low before starting another multiplication cycle, correct?"**
    - Yes your circuit should wait for the run to go low before another calculation is performed, i.e., one multiplication per run.
- **"My state machine does an add in one state, then a shift in the next state, then another add, and so on until the multiplication is done. All these states proceed to the next state regardless of any input values i.e.**
  when A
  next_state = B
  for all the adds and shifts. Does the symbolic state machine in the SV code ensure that exactly one shift will occur in each state, no more and no less, or exactly one add and load operation? Or do we need a signal that says when it's OK to move to the next state."
    - Each state will only be executed for the duration of a single clock cycle, which will only shift your register precisely once if you have raised the shift control signal during that state. You don't need separate signal to monitor the shift.
- **"My state machine is designed to skip ADD states when M=0, but it sometimes goes to those ADD states when M is clearly 0.**

What do I do?"

- Since the actual shift occurs at the end of a SHIFT state, M does not get updated until after the state machine transitions to the next state. To fix this, just define M as B[1] instead of B[0], since B[1] would be M after the shift.

- **"My compilation time is very long on the EWS machines"**
  - Make sure you are not trying to compile across the network. Copy the work directory to a temporary folder on the C: drive, and remember to **move it back and delete your temporary copy after you are done.**