Calculator - Design II

I. Design

This design is an extension to calculator design 1.

The initial design allowed only one command from each of the four ports at a time. All ports needed to wait until the calculator completed execution of the current command before another command could be sent.
In the new design, up to four commands can be sent into the calculator from each of the 4 ports. Hence, (theoretically) the calculator could be working on up to 16 commands at a single time.

This single design change has major implications to the system. Since there are two internal arithmetic pipelines in the calculator (one for add/sub and one for shifts), it is possible for commands to be executed out of order. For example, if the four ports all send in 3 add commands followed by a shift command, the calculator is likely to execute the shift commands prior to the newer (more recently sent) add commands. However, the specification dictates that commands from the same port that use the same pipeline (add/sub or shift) must return in order.

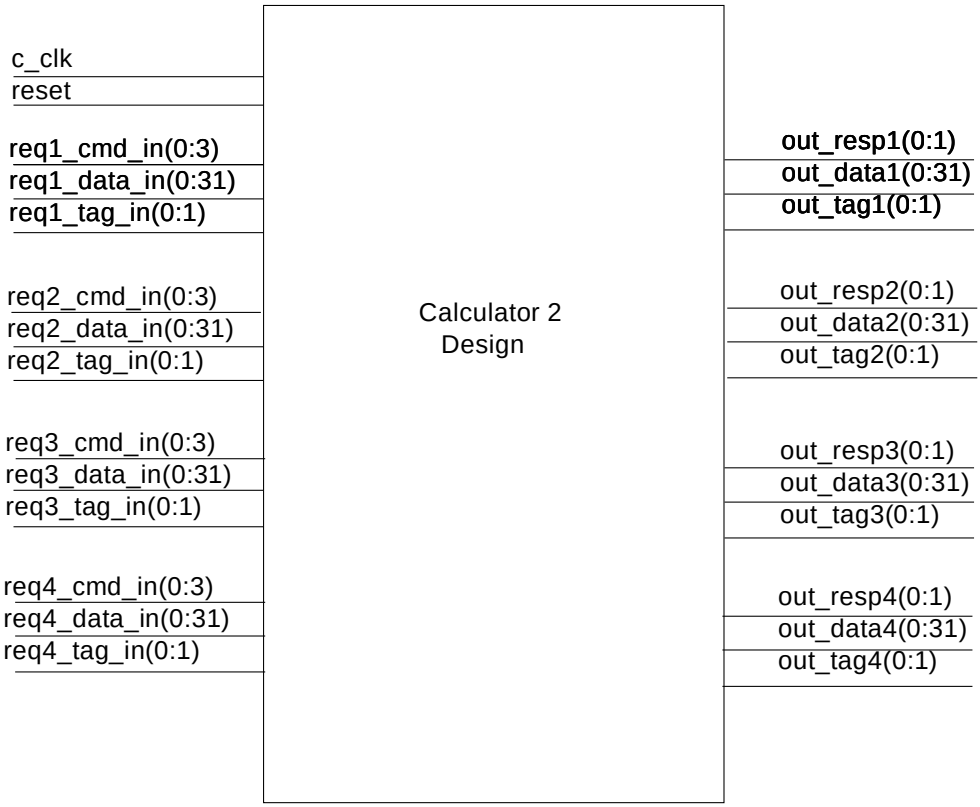
In order to correlate the responses to the correct commands, the specification calls for adding a two bit tag to the input and output protocols. This tag shall be a unique identifier for each of the commands from each port. (Inside the calculator design, the HDL maintains another pair of "internal" tag bits to correlate the command back to the correct port).

II. Problem Statement

This is a new design that contains added complexities. You must verify the correctness of the design through simulation. Use the following information to drive and check the design:

A. Block Diagrams

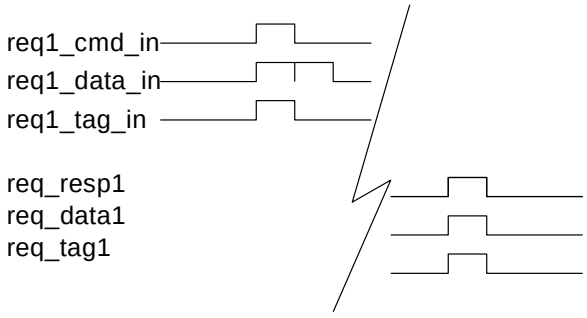
1. Inputs/Outputs of the Calculator



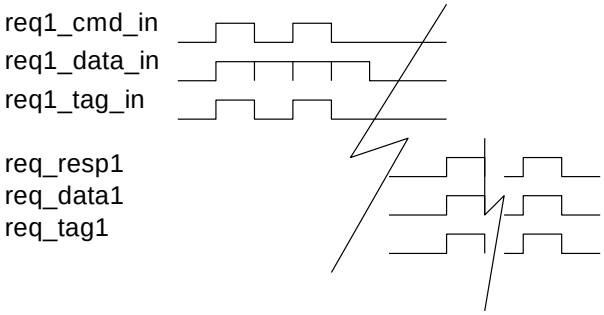
c_clk: c_clk is the main clock.
reset: Needs to be held high for three cycles at the start of the testcase. Must remain low during functional testing. Similarly, all input ports need to be driven low (‘0’b, not “X” or “U”) from the start of simulation.
reqX_cmd_in(0:3): Same definitions as first design. Add (‘0001’b), Subtract (‘0010’b), Shift left (‘0101’b), and Shift right (‘0110’b).
reqX_data_in(0:31): Same definition as first design. The operand data is sent one cycle after another. Operand1 data accompanies command, and operand2 data follows.
reqX_tag_in(0:1): Two bit identifier for each command from the port. Can be reused as soon as the calculator responds to the command.
out_respX(0:1): Same definition as first design. Good response (‘01’b), invalid command or overflow/underflow (‘10’b), and internal error (‘11’b; never happens). ‘00’b on the response line indicates there’s no response from that port on that cycle.
out_dataX(0:31): Same definition as first design. This is the data that accompanies a good response.
out_tagX(0:1): Corresponds to the command tag sent by the requester. Used to identify which command the response if for.

B. Interface Timings

The following diagram represents the valid command input and output. This is the same as the first exercise except that the tag bits are added.



The following diagram represents back-to-back timings on commands.



Responses in above diagram could occur in back-to-back cycles. Responses are not necessarily in order. However, if both requests used the add/sub commands, then they will return in order. Similarly, commands from the same port that share the shift pipeline will return in order.