DESIGN VERIFICATION Practical 1:
Debugging a Calculator Design

This practical requires you to find bugs in a calculator design. The calculator specification is given below. The design HDL code is not available for this practical. You are expected to use a black box verification approach.

For simulation with ModelSim, the calculator design has been provided in the pre-compiled library `calc1_black_box`. The library is available from BlackBoard as a ZIP file. BlackBoard also provides you with instructions on how to set up your workspace with this library. The top-level module of the calculator design is called `calc1`. Please use the example testbench file provided on BlackBoard to check that you have set everything up correctly.

Calculator Specification
Input/Output Specification

The calculator has four commands: add, subtract, shift left and shift right. It can handle (but not process) four requests in parallel. All four requestors use separate input signals. All requestors have equal priority. Figure 1 shows the input output ports of the calculator.
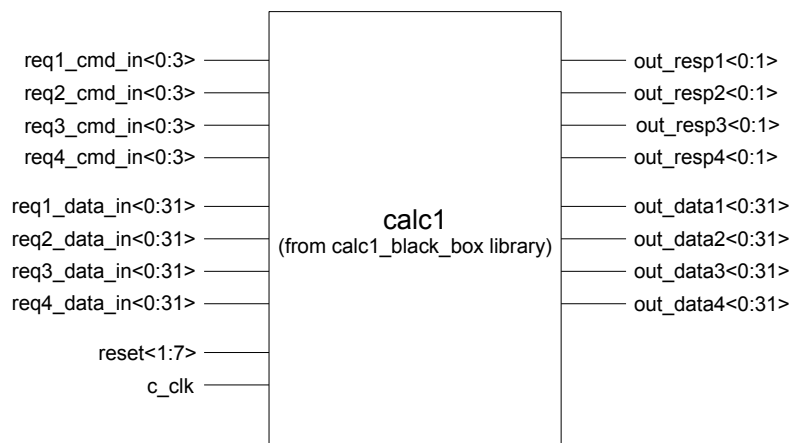

Figure 1: Top-level of Calculator Design

Input commands are coded as follows:
  0   no op
  1   add operand_1 and operand_2
  2   subtract operand_2 from operand_1
  5   shift left operand_1 by operand_2 places
  6   shift right operand_1 by operand_2 places

Input data timing: Operand_1 arrives with the command. Operand_2 arrives on the following clock cycle.

Output response lines encode the following:

0	no response
1	successful operation completion
2	invalid command or overflow/underflow error
3	internal error

Output data timing: Valid result data on output lines accompanies the response on successful completion (i.e. both are driven in the same cycle). Figure 2 depicts the input/output timing on a single port of the calculator. The lightening bolt represents "some number of cycles passing". The data accompanies a successful response signal. A second request from the same port is prohibited until the response from the first command is received.
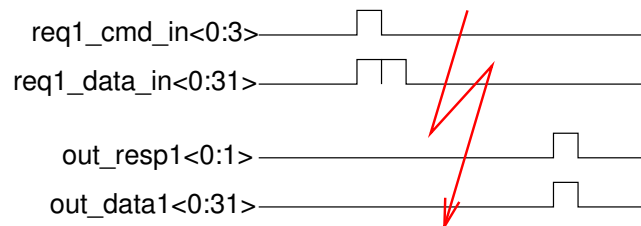

Figure 2: Input/Output Timing of Calculator Design

Clocking: The clock should be toggled when using an event-based simulator.

Priority Specification

The calculator priority logic works on a first come first serve algorithm. It allows for 1 add or subtract at a time and one shift operation at a time.

In other words, if all four ports send an add/subtract at once, then all will be processed one after another (not in parallel). None of the requests will be lost or dropped. It just takes additional cycles to process the 2nd, 3rd, and 4th commands.

Suggested Verification Strategy

- Familiarise yourself with the specification. Make sure you understand what the calculator is supposed to do.

- Plan *what* you want to verify and *how* you are going to do this. Your plan serves as the *specification for your testbench*.

- Design your testbench. Aim to distribute the functionality of your testbench into dedicated modules.

- Implement your testbench including drivers and checkers. The file example_calc1_tb.v contains a number of directed tests and demonstrates how to access the ports of the calculator top module calc1. Note, though, that example_calc1_tb.v is a very poor example for structuring a testbench. (Why?)

- Use a simulator to run your test suite on your testbench and use a waveform viewer only for debugging.

- Be as precise as you can when describing bugs! When you find a bug, try to narrow it down as much as you can. For example, it is more helpful if you tell a designer that: "Subtraction does not work on bits 8 and 9 (counted from the least significant bit upwards)." than if you say: "Subtraction does not work." or "It does not do 100000 - 6 correctly."

Remember that your testbench needs to be generic; it should not be specific to the version of the calculator you are working with. This means that, even if we were to change the implementation of the calculator, e.g. next week and again the week after, then your testbench should still be effective at finding bugs in any new version of the calculator.