Purpose

The purpose of this lab is to: 1) Behaviorally describe using multiple processes a CPU performing a certain computation with data obtained from an external RAM. 2) Describe a fully responsive handshaking protocol between a CPU-memory interface and the RAM (Memory) unit. 3) Design and behaviorally describe a simple pipelining approach of performing the computation on the current data while the next one is being loaded from memory.

Design Problem

A behavioral description of the 3 modules, CPU, CPU-Mem Interface (CMI), and Memory (Mem) shown in Fig. 1 needs to be developed with one process per module. The following are the functions of the different modules.

Module Functionality

1. CPU:
   (a) It starts its processing on getting an external go signal going high.
   (b) It performs the computation

   \[
   \sum_{i=1}^{n} a_i^2
   \]

   where the \( a_i \)'s represent unsigned (i.e., positive) 32-bit integers, and are stored consecutively in the RAM store that is part of the memory module. The \( a_i \)'s need to be declared as 32-bit std_logic vectors. \( n \) needs to be a generic size parameter of your design.
   (c) It initializes and then updates an address register AR that points to the next \( a_i \)'s address in RAM. It also has a data register DR. Both AR and DR are accessible to both the CPU and Mem modules. Such shared objects like registers can be implemented using global variables declared in the architecture body (as opposed to within a process). This issue will be explained in greater detail later,
Figure 1: Three modules and inter-module communication signals

Figure 2: A 2-stage pipeline
(d) It raises the read or write signal to the CMI that informs it that a read or write, respectively needs to be performed. The read needed has the functionality

\[ DR \leftarrow RAM[AR] \]

and the write needed has the functionality

\[ RAM[AR] \leftarrow DR \]

The read (write) signal is held high until the CPU finds that the DR has valid (invalid) data indicated by an extra valid bit associated with the DR that is only written to by CMI. Thus the signaling between the CPU and CMI is of the partially responsive variety.

(e) It requests reads to be performed such that the next data element \( a_{i+1} \) is being loaded from RAM while the aforementioned computation on the current data \( a_i \) is being performed. This results in a pipelined processing of the data; an abstract model is shown in Fig. 2.

(f) At the end of the computation, it writes the final result in DR, and requests the CMI to store it in RAM at an appropriate address.

2. CMI: Processes read/write requests from the CPU by fully responsive communication (each module looks for the other’s control signal to be raised and then lowered before proceeding to the next transaction) with the Mem module.

Some basic signals between CMI and Mem are shown in Fig. 1. If any others are needed, add them, explicitly state what they are and their purpose, and show them in a similar block diagram (as Fig. 1) in your report.

3. Mem:

(a) On an external initialize high signal, it initializes the first \( n \) locations of the RAM with the \( a_i \) values as follows: \( a_1 = 10 \). For the first \( n/2 \) data items, it then repeats the pattern of alternately adding 2, 5, or 8 to \( a_i \) to generate \( a_{i+1} \). Thus \( a_2 = a_1 + 2, a_3 = a_2 + 5, a_4 = a_3 + 8, a_5 = a_4 + 2, a_6 = a_5 + 5 \), and so on. For \( i = n/2 + 1 \) to \( n \), \( a_i \) is generated by alternately subtracting 3 or 7 in a repeating pattern from \( a_{i-1} \). Thus \( a_{n/2+1} = a_{n/2} - 3, a_{n/2+2} = a_{n/2+1} - 7, a_{n/2+3} = a_{n/2+2} - 3 \), and so on.

(b) Processes read/write requests from the CMI using fully responsive signaling with it using control signals and data buses shown in Fig. 1.

(c) Sets the result output to the final result after writing it to the RAM, when it receives an indication from the CMI that this is the final result that it is being asked to write.
Other Issues

1. Global variables visible to and used by multiple processes are declared in the architecture body using the additional `shared` keyword as exemplified below (with acknowledgements to the VHDL tutorial at http://www.vhdl-online.de/ [slide 109]).

```vhdl
architecture BEHAVE of SHARED is
shared variable S : integer;
begin
    process (A, B)
    begin
        S := A + B;
    end process;

    process (A, B)
    begin
        S := A - B;
    end process;
end BEHAVE;
```

In the above example, care needs to be taken that both processes do not write to S at the same time, otherwise the results are non-deterministic. Similarly, for your design you need to take care that DR is not written by the CPU and CMI processes at the same time—this may not need to be done explicitly, if you can ascertain/prove that for your design, the way the processing is structured, this cannot happen; otherwise, you will need to explicitly design for ensuring this.

2. Describe your design in VHDL so that the following delays are incurred.

   (a) *read/write* signal propagation delay is 25 ps.
   (b) Register read/write takes 200 ps.
   (c) RAM read/write takes 1 ns.
   (d) Signal propagation delay (control or data signals/buses) between CMI and Mem modules is 2 ns.
   (e) Square computation takes 4 ns.
   (f) Add computation takes 1 ns.
(g) Note, for example, that a signal assignment such as \( address \leftarrow AR \); incurs two delays, one to access AR and the second is the propagation delay on the address bus.

(h) If there are other delay parameters that you need to use, let me know and you can assume suitable values for them (in line with the above delays). State these delay parameters clearly in your report.

**Analysis**

Ignoring the RAM initialization phase, theoretically analyze the delay it will take your design to produce the result of the CPU’s computation and display it on the output result line. Note that your design is supposed to process the data in a pipelined fashion, and the analysis should thus be for a pipelined processing of the data.

**Test Bench**

Describe a simple TB in which the above design is instantiated, and which starts its processing by sending a high initialize and go signals in the proper order and with the appropriate gap between them. These signals should also be lowered after an appropriate time period.

**Simulation**

Simulate your design for \( n = 16, 32, 64, 128 \) and 256. For each \( n \), show correctness of the final result, and the delay in obtaining it starting from when the go signal is asserted. Is this simulation delay consistent with your analyzed delay? If not, explain why not. Finally, plot the delays (y-axis) for each \( n \) (x-axis). Curve fit to the closest linear function (if any), determine its slope, and relate the slope to a relevant metric in your delay analysis, explaining the relation.