Lab 5 Practice: Random Memory Access

Objective

1 Getting familiar with the memory usage.

Action Items

In this exercise, you will explore the concept of random memory access to gain a deeper understanding of how data can be retrieved and manipulated efficiently. Your task is to implement a solution that demonstrates your ability to handle memory accesses at arbitrary locations, which is crucial for optimizing performance in various algorithms and systems.

A. lab5_practice.v

Design an accumulator that performs the summation of data stored in a memory array of six 8-bit entries.

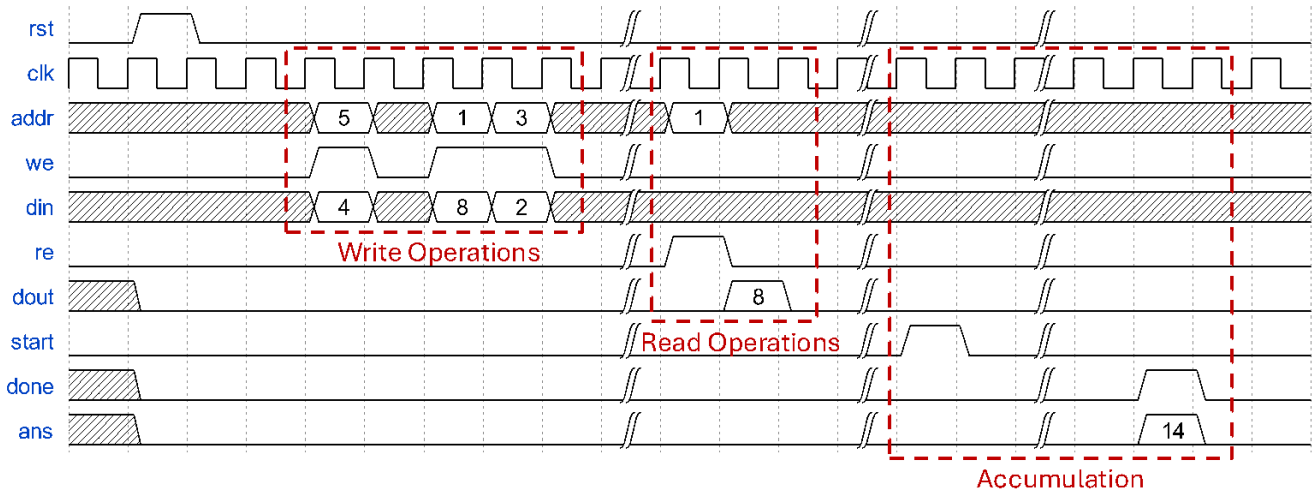
IO Specifications:

- Inputs: clk, rst, we, din, addr, re, start
  a. **rst** (synchronous positive reset): the ans would be 0 if **rst**==1'b1
  b. **addr** (address): the location in memory that you are going to write or read
  c. **we** (write enable signal): write the data-in (**din**) into the memory when **we**==1'b1
  d. **din** (data-in): the value you are going to write into the **addr** position of the memory
  e. **re** (read enable): read the data from the memory when **re**==1'b1
  f. **start** (control signal): trigger the accumulator to sum up the data in the memory
- Output: ans
  a. **dout** (data-out): the value read from the **addr** position of the memory
  b. **done** (control signal): indicate the completion of the summation
  c. **ans** (answer): the accumulated answer

Functionality:

- The memory of six 8-bit entries can be modeled as
  ```
  reg [7:0] memory [0:5];
  ```
  in Verilog.

- Upon reset, the memory content and accumulator's outputs are cleared to zero. To reset the memory content, you can utilize a for-loop statement in your Verilog code.
- Read and write operations must be synchronized with the positive clock edges.
- Create your own testbench to write the data into the memory at arbitrary addresses. Additionally, you can overwrite the existing values.
- Read out the values stored in the memory to verify correctness.
- Refer to the following timing diagram as an example.
  Note that the slashed regions in the diagram represent **don't-care values**, not **unknown values**, meaning the logic levels at these points are irrelevant to the behavior of the design.
- After the **start** is asserted (i.e., start==1'b1) for one clock cycle, the accumulator begins to sum the values stored in the memory from **address 0** to **address 5 sequentially**. Once the summation is complete, the **done** signal will be asserted (i.e., done==1'b1) for one clock cycle. Simultaneously, the accumulated result will be shown on the **ans** output.



You must use the following template for your design:

```
module lab5_practice (
          input wire clk,
          input wire rst,
          input wire [2:0] addr,
          input wire we,
          input wire [7:0] din,
          input wire re,
          input wire start,
          output reg [7:0] dout,
          output reg done,
```

```
        output reg [7:0] ans
);
    // Add your design here
    // Note that you are free to adjust the IO's data type
endmodule
```

Attention

- ✓ If you create several modules for your design, merge them all into one Verilog file.
- ✓ This practice helps you with the following basic lab, please make sure you completely understand it.
- ✓ Feel free to ask questions about the specification on the EECLASS forum.