ECE 464 / ECE 564
Homework 6
Revision: 1.2

On-line turn-in. Check out the submission instructions below. These will be checked using Code Comparison tools. If your code is substantially similar to someone else's you will both receive an academic violation for cheating.

Question 1

You are to design a module that accumulates floating point numbers in an array. The array is stored in memory Fig. 1 show the layout of memory. The array size(N) value is stored in address 0. The contents of the array are stored from address 1 to N, and the result of the accumulation needs to be stored at address N+1. Floating pointing numbers are not commutative, so the order of accumulation should be from 0 to N-1.

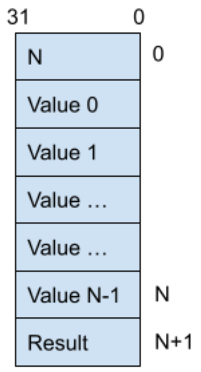

Fig. 1 SRAM Contents

The I/O to the module is as follows:

```
//-----------------------------------------------------------------
//System signals;
  input wire reset_n;     \\ synchronous reset - active low
  input wire clk;         \\ Clock

//-----------------------------------------------------------------
//Control signals
  input wire dut_valid;  // synchronous valid – active high
  output wire dut_ready; // synchronous ready – active high

//-----------------------------------------------------------------
//SRAM interface
  output wire         sram_write_enable;  // sync ena active high
  output wire [15:0]  sram_write_address; // sync waddr data
  output wire [31:0]  sram_write_data;    // sync wdata data
  output wire [15:0]  sram_read_address;  // sync raddr data
  input  wire [31:0]  sram_read_data;     // sync rdata data
```

System signals:

- *reset_n* is used to reset the logic into a known state,
- *clk* is used to drive the flop logic in the design.

Control signals:

- *dut_valid:* used as part of a hand shack between the test fixture and the dut. Valid is used to signal that a valid input can be computed from the SRAM.
- *dut_ready:* used to signal that the dut is ready to receive new input from the SRAM.
- Together these two signals tell the test fixture the state of the dut. So, the dut should assert *dut_ready* on reset and wait for the *dut_valid* to be asserted. Once *dut_valid* is asserted by the test fixture, the dut should set *dut_ready* to low and can start reading from the SRAM. Dut should hold the *dut_ready* low until it has populated the result value in the SRAM. Once the the result is in the SRAM the *dut_ready* will be asserted high, signaling that the result is valid, and is ready to be read from SRAM. Fig 2 shows the expected behavior.

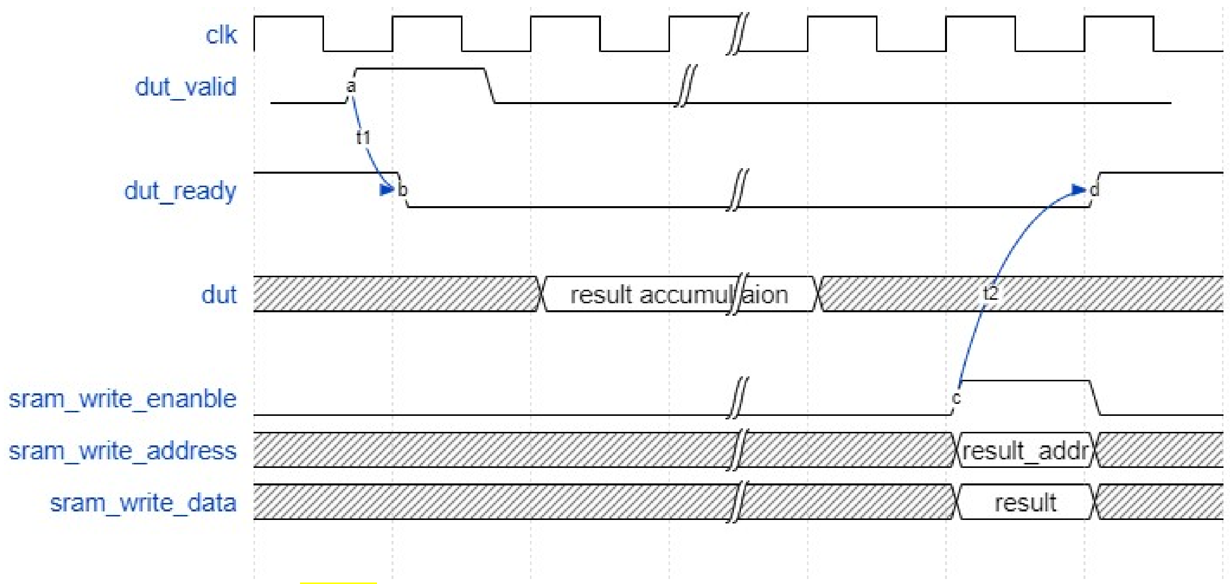

Fig 2. Test fixture and DUT handshake behavior

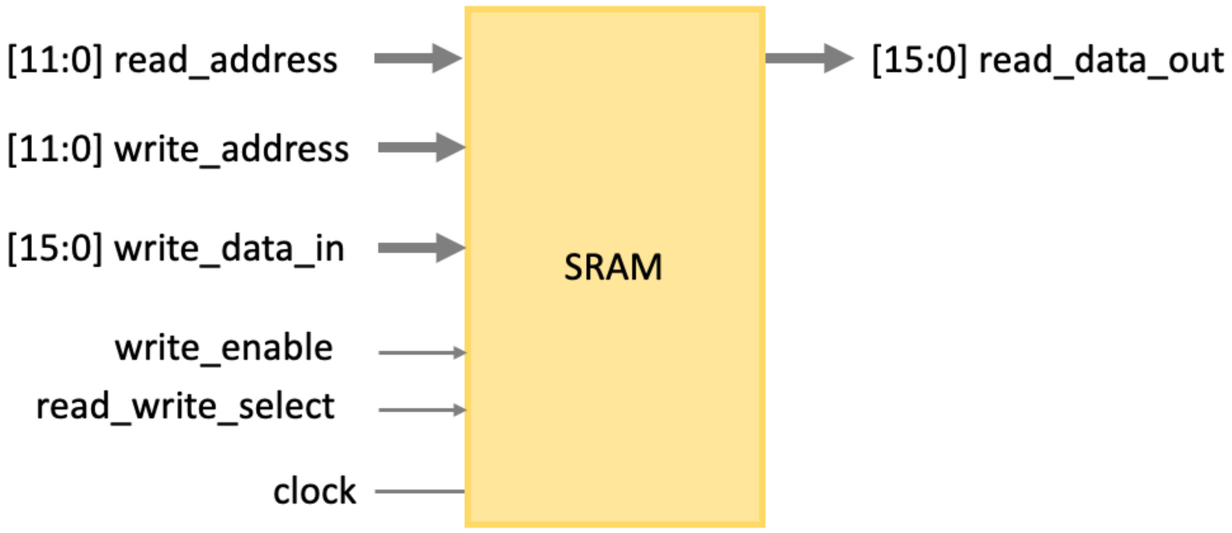

Fig 3. SRAM interface ports

The SRAM is word addressable and has a one cycle delay between address and data. When writing to the SRAM, you would have to set the "write_enable" to high. The SRAM will write the data in the next cycle. "read_write_select" is not used for this implementation.

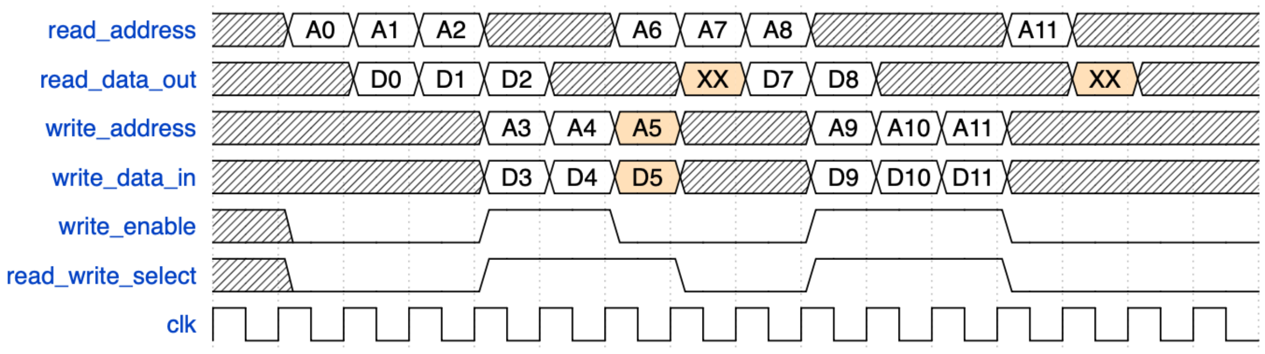

Fig 4. SRAM timing behavior

As shown in the Fig 4, since "write_enable" is set to low when A5 and D5 is on the write bus, D5 will not be written to the SRAM. Also, because "read_write_select" is set to high, the read request for A6 will not be valid.

Note that the SRAM cannot handle consecutive read after write (RAW) to the same address (shown as A11 and D11 in the timing diagram). You would have to either manage the timing of your access, or write the data forwarding mechanism yourself. As long as the read and write address are different, the request can be pipelined.

Important Notes:

1. This HW is designed to help you get a start on the project.
2. Please read the README and look at the dut.sv file to understand how to build the floating-point units.
3. Since you need to incorporate designware the Questa versions of Modelsim won't work.

Design, verify, synthesize a module that meets these specifications. **Use at least one coding feature unique to System Verilog.**

Submission Instruction:

- **Project Verilog and synthesis files.** Submitted electronically on the date indicated in the class schedule. Please turn in the following:
  - All Verilog files AS ONE FILE
  - Modelsim simulation results file showing correct functionality. Logs from 'HW6/run/logs/*.log'
  - Synopsys view_command.log file from complete synthesis run
- **Project Report.** Complete report to be turned in electronically with project files. It must follow the format attached. There is a 10% penalty for not following the format.

[50 points]

Report First Page. Must match this format (Title) – attach other pages as needed

StudentID. Name:

Delay (ns to run provided example). Clock period: # cycles":	Area: (um^2) Logic: Memory: N/A	1/(delay.area) ($ns^{-1}.um^{-2}$) :

FSM Diagram:

Grading Rubric

Element	**Standard**	**Grade**
Report	FSM Diagram; simulation/synthesis details.	**10/10**
Missing one of the elements above	**5/10**	
Deficient in all elements above	**0/10**	
Code	Looks correct in grammar and matches drawing; runs correctly on provided data set.	**20/20**
Missing one of the elements above	**10/20**	
Deficient in all elements above	**0/20**	
Mystery run	Mystery data set runs correctly.	**10/10**
Shows incorrect behavior	**0/10**	
Timing	Meets setup and hold	**5/5**
Meets only one	**3/5**	
Does not meet either setup or hold	**0/5**	
Area	Reported	**5/5**
Not reported	**0/5**	
Total	**Out of 50**	