ECE 464 / ECE 564
Mini-project and HW 6
Revision: 1.0

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

You will be designing a floating point multiply accumulate DesignWare module to perform matrix multiplication. The input matrix array is stored in a static random access memory (SRAM) as shown in Figure 1. An example is given below. (The actual matrix size will vary).

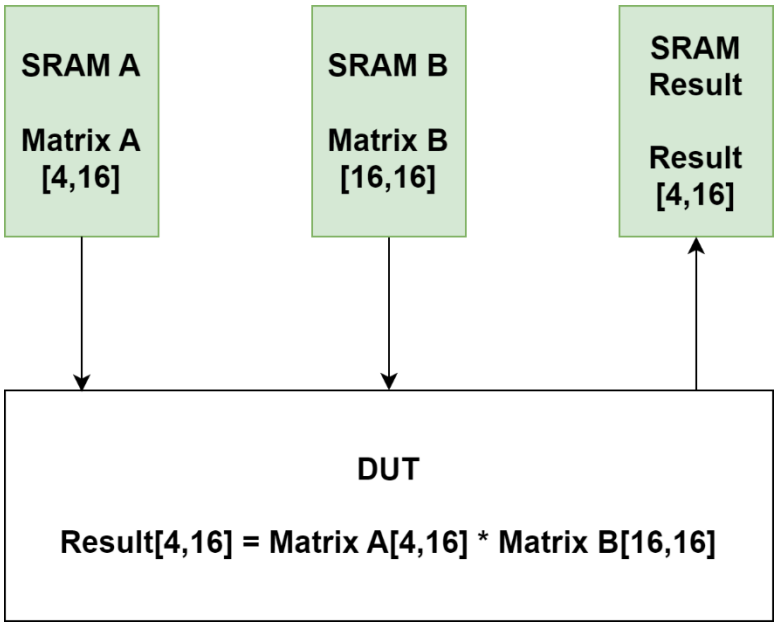

Figure 1: SRAM connection to the Design Under Test (DUT) and the memory allocation.

The matrix multiplication is performed on Matrix A (SRAM A), Matrix B (SRAM B) and the results will be stored in SRAM Result. The equation below, shows the matrix multiplication performed:

$$\begin{bmatrix} C_1 & C_2 & C_3 & C_4 \\ C_5 & C_6 & C_7 & C_8 \\ C_9 & C_{10} & C_{11} & C_{12} \\ C_{13} & C_{14} & C_{15} & C_{16} \end{bmatrix} = \begin{bmatrix} A_1 & A_2 & A_3 & A_4 \\ A_5 & A_6 & A_7 & A_8 \end{bmatrix} * \begin{bmatrix} B_1 & B_5 & B_9 & B_{13} \\ B_2 & B_6 & B_{10} & B_{14} \\ B_3 & B_7 & B_{11} & B_{15} \\ B_4 & B_8 & B_{12} & B_{16} \end{bmatrix} \quad \text{(Eq: 1)}$$

As regular matrix multiplication you will multiply and accumulate each row element of matrix A with column elements of matrix B. The expression below shows the example computation:

$$C_1 = (A_1 * B_1) + (A_2 * B_2) + (A_3 * B_3) + (A_4 * B_4)$$

$$C_2 = (A_1 * B_5) + (A_2 * B_6) + (A_3 * B_7) + (A_4 * B_8)$$

System signals:

- *reset_n* is used to reset the logic into a known state,
- *clk* is used to drive the flip-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 values in the result SRAM. Once the results are stored 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.

There is no DUT signal. This just indicates when computation is proceeding.

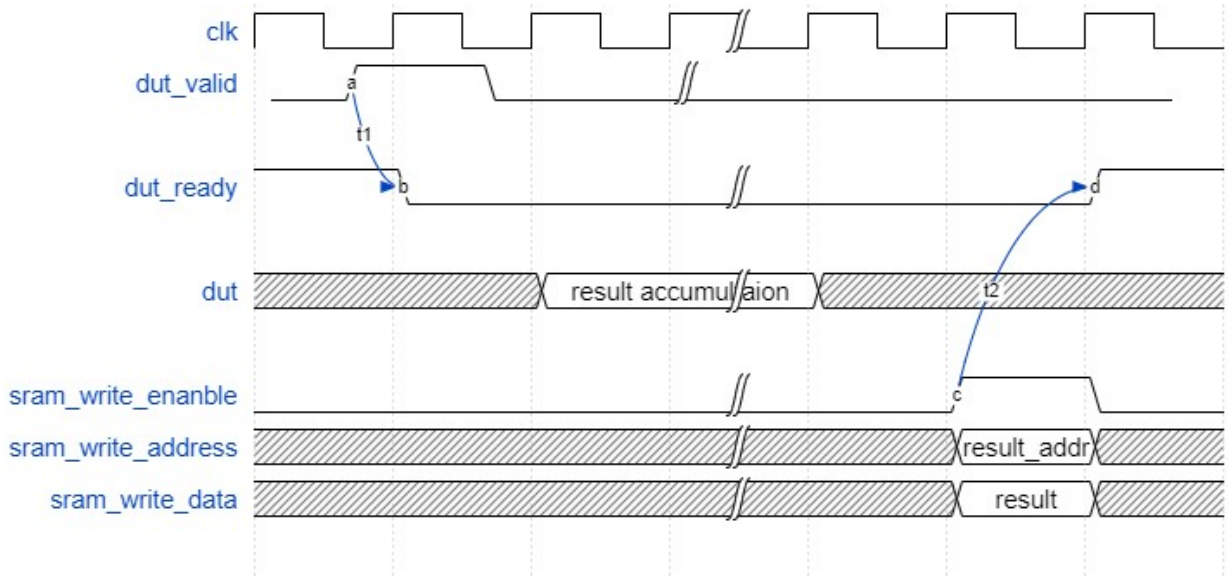

Fig 2. Test fixture and DUT handshake behavior

SRAM contents:

The SRAM holds the 32-bit data in each address. The table below shows the memory mapping of various SRAM addresses. The color schemes corresponding to Equation 1. The input SRAM's contains the input matrix dimension at address 12'h00, while the matrix data from 12'h01 onwards.

All data is 32-bit signed. Ignore overflow.

SRAM A: Address	SRAM A: Content [31:0]
12'h00	[31:16] – Number of matrix A rows, [15:0] – Number of matrix A columns
12'h01	A_1
12'h02	A_2
.	.
.	.
12'h08	A_8

SRAM B: Address	SRAM B: Content [31:0]
12'h00	[31:16] – Number of matrix B rows, [15:0] – Number of matrix B columns
12'h01	B_1
12'h02	B_2
.	.
.	.
12'h10	B_{16}

SRAM Result: Address	SRAM Result: Content [31:0]
12'h00	C_1
12'h01	C_2
.	.
.	.
12'h07	C_8

SRAM:

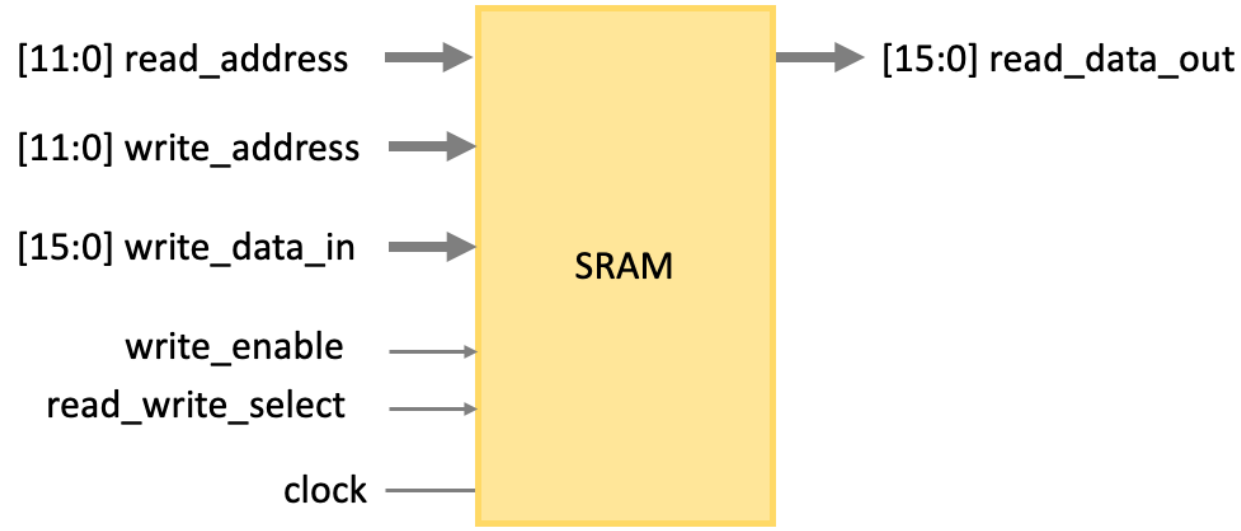

Fig 3. SRAM interface ports

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.

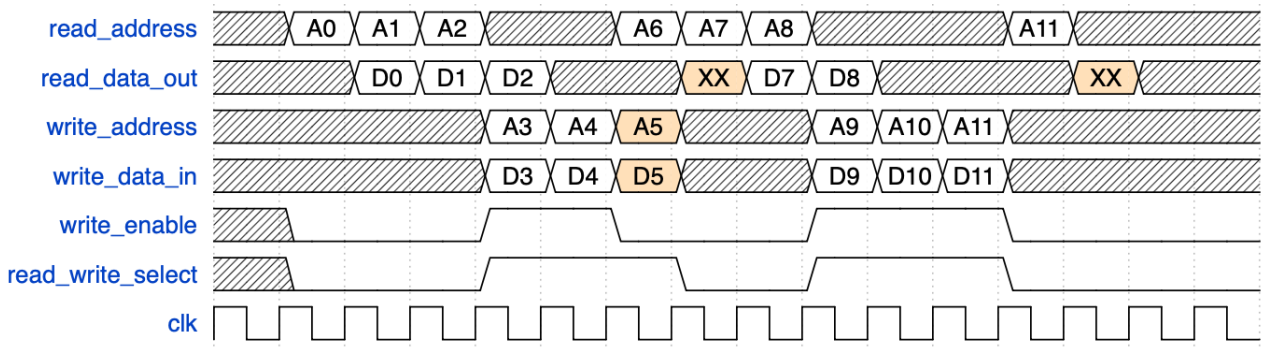

Fig 4. SRAM timing behavior

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 mini-project 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 and needs to be connected to Grendel servers to access the libraries.
4. Design should not have any major/minor synthesis errors pointed out in the Standard Class Tutorial (Appendix C). This includes but not limited to latches, wired-OR, combination feedback, etc.

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
    - Zipped modelsim simulation results file showing correct functionality. Logs from 'mini_proj/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]