Module Name: my_calc

Design Engineers: Eldridge Surianto

Verification Engineers: Eldridge Surianto

Test Bench File: calc_tb_top.sv

Functionality	Basic Computation
□ Normal addition with no overflow	The testbench runs a "normal addition" scenario by calling calc_driver_h.start_calc(0, 4, 5, 9). This tests the core functionality of reading a range of data from SRAM A and B, adding them, and writing the results. The scoreboard (calc_sb_h) verifies the correctness of the final values in SRAM.
□ Normal Addition with overflow	The testbench explicitly writes 32'hFFFFFFFF to addresses in both SRAMs and then initiates a calculation. This is designed to trigger an arithmetic overflow. The scoreboard checks if the wrapped-around result is calculated and stored correctly.

Functionality	Edge Cases
□ Single Address Operation	The testbench calls calc_driver_h.start_calc(20, 20, 21, 21), where the start and end addresses are the same. This verifies that the DUT correctly handles a calculation involving a single memory location.
Overlapping Read/Write Addresses	The test calc_driver_h.start_calc(10, 15, 12, 17) is executed. This case checks for potential data corruption by having the write operation start in the middle of the read address range, ensuring the DUT reads all values before they are overwritten.

Functionality	Data Flow and SRAM Integrity
□ Data Read	Data read integrity is implicitly verified in every test case. The calc_driver_h initializes SRAMs with known values. The monitor (calc_monitor_h) captures the transaction data, and the scoreboard (calc_sb_h) compares the final written data against the expected sum, which can only be correct if the initial data was read properly.
□ Data Write	Data write integrity is the primary check of the scoreboard (calc_sb_h). After each calculation (start_calc), the scoreboard compares the values in the DUT's SRAM against a golden model it maintains, ensuring that the results were written to the correct addresses with the correct data.

Functionality	Timing & Reset Tests (Assertion)
□ Reset functionality	A dedicated task reset_in_state forces the FSM into specific states (S_READ, S_ADD, S_WRITE, S_END) and then applies a reset. It then checks if the FSM correctly returns to S_IDLE.
□ Verify the buffer_loc toggling logic.	**Assertion:** The property BUFFER_LOC_TOGGLES is defined to `assert property (@(posedge clk) (state == S_ADD)
□ Verify the valid input addr	The property VALID_INPUT_ADDRESS is defined to `assert property (@(posedge clk) (state == S_READ)

Functionality	Random Constrained Test (Coverage)
□ give random input	Verify the output. Overall Coverage for DUT >= 96%

Functionality	FSM Coverage
□ FSM coverage	Coverage reaches 100%