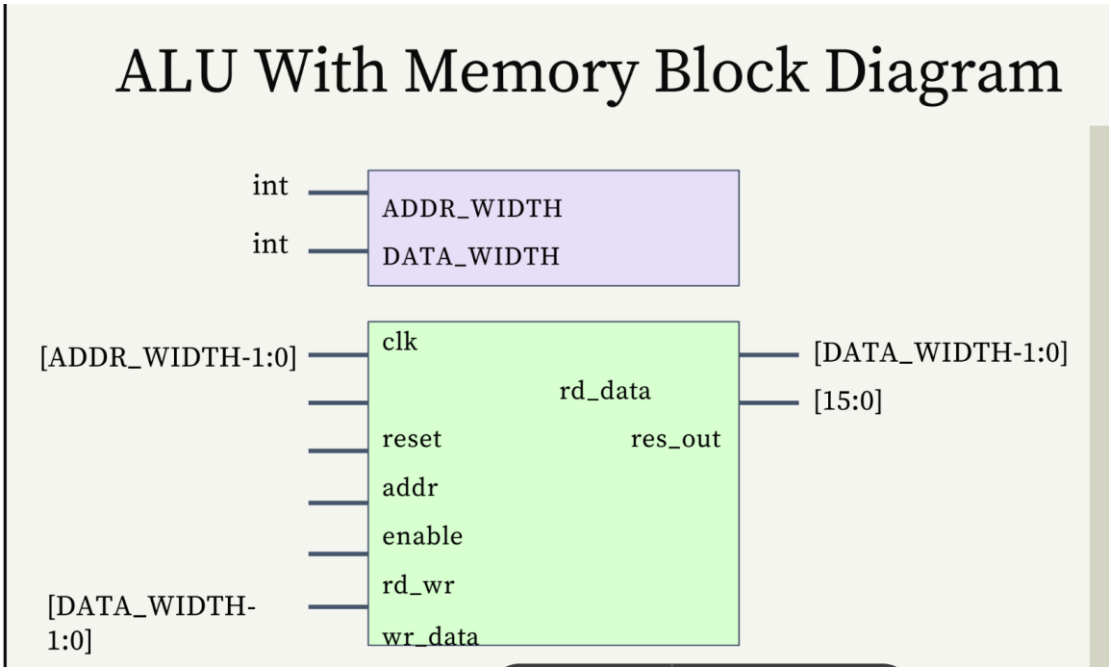
ALU with Memory



Here's the rewritten version of alu with memory verification plan:

Verification Plan:

-The plan outlines the scenarios to be tested, including write and read operations on specific memory addresses.

-Single Location Test: Write data to a memory address, then read from the same address, ensuring the read data matches the written data.

-Full Memory Test: Perform write and read operations across all memory locations. With a 3-bit address width, verify addresses from `3'b000` to `3'b111.`

-Default Value Check: Before performing any writes, read all memory locations to confirm the default value is `hFF.`

-Reset During Operations: Trigger a reset while performing write/read operations and confirm that all memory locations return to their default value (`hFF`) after the reset.

-Objective: Verify correct behavior for basic ALU operations (e.g., addition, subtraction).

-Apply specific inputs to the ALU for addition, subtraction, multiplication, and division operations.

-Compare the output with the expected result.

-Perform operations that generate zero results.

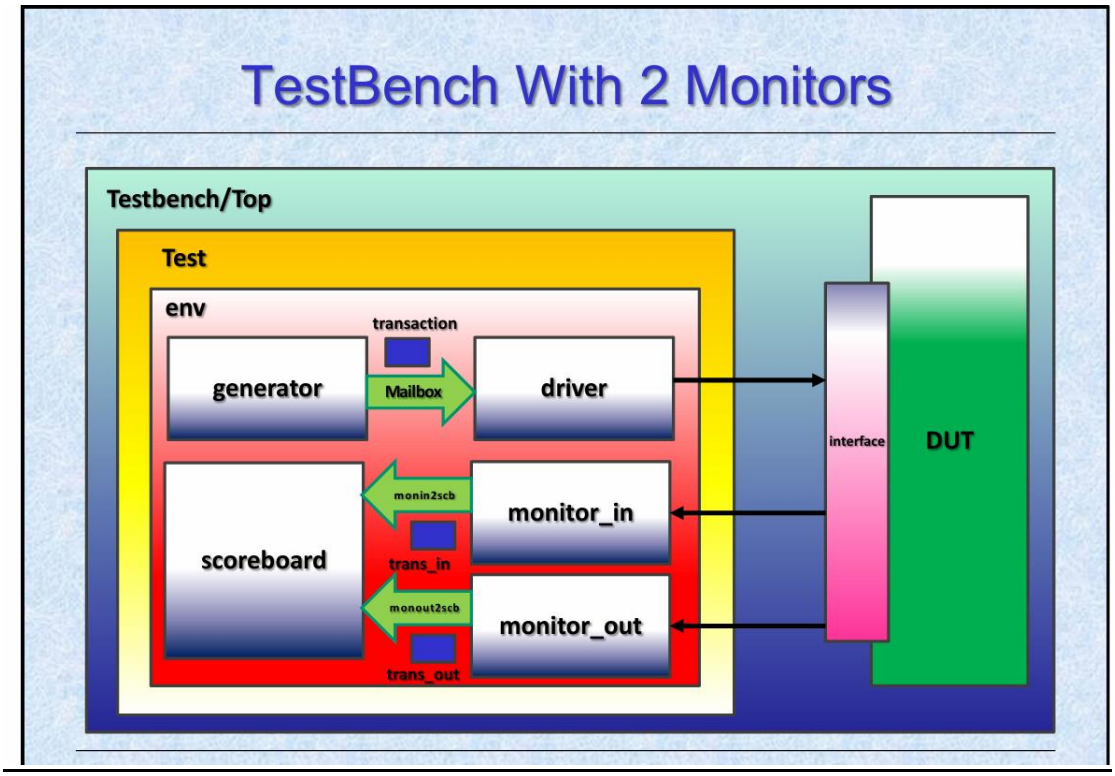
Division by Zero Handling:

-Objective: Ensure the ALU handles division by zero correctly.

Attempt a division by zero.

-Verify the ALU outputs the appropriate error code or handles the case as per the design (e.g., output a specific value like 16'hDEAD).

Class explanation



1-DUT (Device Under Test):** The design consists of two modules: a memory and an ALU.

The memory module has 8 locations, each 8 bits wide, with inputs `clk`, `rst`, `enable`, `rd_wr`, `addr`, and `wr_data`, and an output `rd_data`.

It also has inputs `A_reg`, `B_reg`, `Oper`, and `Execute`. On reset, the memory initializes all locations to `8'hFF`.

When `rd_wr` is low, data is written to the memory at the specified address; when `rd_wr` is high, data is read from the memory.

The design also includes an ALU module with four inputs: `A_reg`, `B_reg`, `Execute`, and `Oper`, and one output `Alu_resout`.

The inputs `A_reg`, `B_reg`, `Oper`, and `Execute` are updated in the memory block when `enable` is 1 and a read operation is performed.

These inputs are updated based on the value of `mem[addr]` and then passed to the ALU module.

In the ALU, based on these inputs, the operation to perform is selected, and the result (`res_out`) is updated accordingly.

2. Interface: The `inf` interface bundles the memory module's signals: `clk`, `rst`, `enable`, `rd_wr`, `addr`, `wr_data`, and `rd_data`, res_out. It simplifies connecting the testbench to the DUT.

3. Transaction Class: This represents a transaction for the memory, containing randomized fields for `wr_data`, `addr`, `enable`, and `rd_wr`. It also captures the `rd_data` and , res_out from the DUT during read operations.

4. Generator: The generator produces random memory transactions and sends them to the driver through a mailbox. It specifies whether to read or write, along with the data and address to be used.

5. Driver: The driver receives transactions from the generator via the mailbox, then drives the corresponding values (e.g., `wr_data`, `addr`, `enable`, `rd_wr`) onto the interface to interact with the DUT.

6. Environment: The environment class coordinates all verification components—generator, driver, monitors, and scoreboard. It handles the virtual interface and mailboxes to facilitate communication between these components.

7. Monitor In: This monitor samples the input signals from the virtual interface, capturing the address, `wr_data`, `enable`, and `rd_wr` fields. It sends the collected transaction data to the scoreboard through a mailbox for input verification.

8. Monitor Out:The monitor_out captures the `rd_data` and , res_out output from the memory when `rd_wr` is set for a read operation. It sends the transaction to the scoreboard for comparison with expected results.

9. Test: The test instantiates the environment, sets the number of transactions (100 in this case), and runs the simulation by triggering the generator, driver, and monitors.

10. Top Module: The top module connects the DUT (memory) and the verification environment. It generates the clock and reset signals and sets up the simulation environment for verification.

11. Scoreboard: The scoreboard compares the expected results with the actual results from the DUT. It checks if the read data matches the expected data for each address and flags any discrepancies. It keeps track of the number of transactions processed and displays results.

In the scoreboard we have a reference model for the memory that works the same as the DUT memory , based on these inputs (trans_in) it updates .

Also we have ALU reference model that works the same as DUT ALU .

If enable==1 and it is write case , we write in the mem_ref[trans_in.addr] , if it read case , we check if the mem_ref[trans_in.addr] is the same as the trans_out.rd_data (the output from the DUT) , and we update the ALU inputs , AND check the output of the reference alu with the trans out res out (the result from the actual design) .