EC22B1004 - P Kaarthick Natesh
05/01/2026 - 25EC5013

High Level Verification using UVM

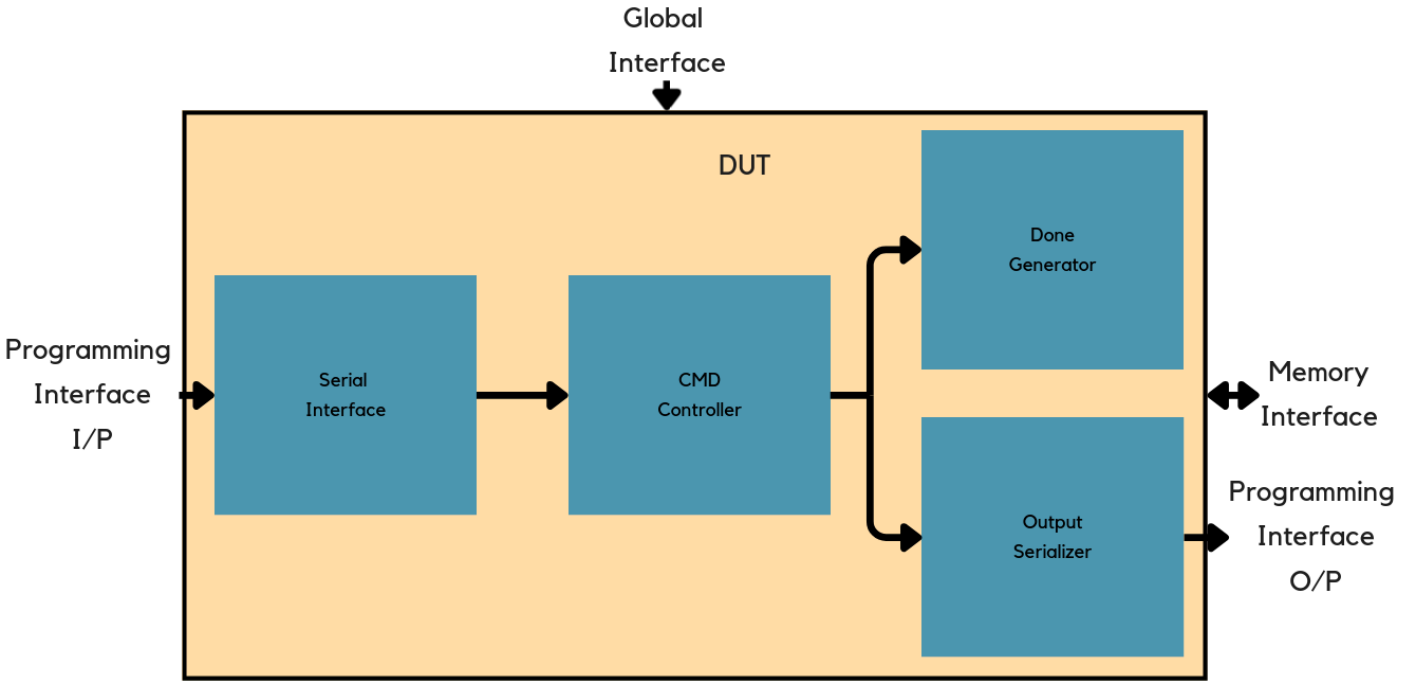
LAB-1: Setting up simple testbench in verilog for a nibble CPU DUT

Objective:

- To understand DUT specification, design and RTL
- To develop a simple testbench to verify basic operations for nibble CPU DUT
- To understand how to instantiate DUT in testbench
- To generate stimulus to DUT to perform add and multiplication operation

Tasks:

- Understanding the DUT



DUT - Nibble CPU System

- **Purpose**
  - Execute memory-mapped instructions.
  - Support 32-bit ADD and MUL.
  - Serialize results as 4-bit nibbles.
- **Top Module - `cpu_top`**

`cpu_top.sv` Interfaces

- **Global Interface:** `clk`, `rst_n`
- **Programming Interface:**

    - `[3:0] prog_nibble_in`
    - `prog_nibble_in_valid`
    - `[3:0] prog_nibble_out`
    - `prog_out_valid`
    - `prog_done`
- **Memory Interface:**
    - `[11:0] mem_addr`
    - `[31:0] mem_wdata`
    - `[31:0] mem_rdata`
    - `mem_wen`
    - `mem_cen`

Internal Modules and Their Roles

`serial_interface.sv`

- **Function:** Input deserialization
- **Operation:**
    - Collects 4-bit nibbles.
    - Assembles 52-bit instruction.
- **Output:**
    - `[11:0] frame_addr` - Address to be modified or read
    - `[7:0] frame_cmd` - Command for CPU
    - `[31:0] frame_wdata` - Data to be written
    - `frame_valid`

`cmd_controller.sv`

- **Function:**
    - Central control FSM.
    - Decodes commands.
    - Sequences memory accesses.
    - Performs arithmetic/logical operations.
    - Manages result, errors, and serializer handshaking.
- **Input:**
    - `frame_valid` - Command available
    - `frame_addr[11:0]` - Base memory address
    - `frame_cmd[7:0]` - Opcode
    - `frame_wdata[31:0]` - Write data
    - `frame_ready` - Controller idle / ready
- **Result / Status Interface**
    - `cmd_start` - Command accepted
    - `cmd_busy` - Controller active
    - `cmd_result[31:0]` - Operation result
    - `cmd_result_valid` - Result valid
    - `cmd_error` - Error occurred
    - `cmd_error_type[1:0]` - Error classification
    - `serializer_busy` - Downstream back-pressure
- **Memory Interface**

- `mem_addr[11:0]` - Memory address
- `mem_wdata[31:0]` - Memory write data
- `mem_rdata[31:0]` - Memory read data
- `mem_wen` - Write enable
- `mem_cen` - Chip enable

`output_serializer.sv`

- **Function:**
  - Serializes 32-bit parallel data into 4-bit nibbles.
  - Outputs nibbles sequentially (LSB first).
  - Controls nibble sequencing and valid signaling.
  - Indicates busy status during serialization.
- **Input:**
  - `clk` - System clock
  - `rst_n` - Active-low reset
  - `data_in`[31:0] - Parallel data to be serialized
  - `data_valid` - Start serialization request
  - `cmd_busy` - Controller busy status
- **Output:**
  - `serial_out`[3:0] - Serialized output nibble
  - `serial_valid` - Output nibble valid
  - `serializer_busy` - Serializer busy indicator

`done_generator`.sv

- **Function:**
  - Detects completion of command execution.
  - Generates program-done pulse.
  - Differentiates success and error completion.
  - Controls pulse width based on error condition.
- **Input:**
  - `clk` - System clock
  - `rst_n` - Active-low reset
  - `cmd_start` - Command start indicator
  - `cmd_busy` - Command execution status
  - `cmd_error` - Error flag from controller
  - `cmd_error_type`[1:0] - Error classification
- **Output:**
  - `prog_done` - Program completion indicator

Testbench -

`` `timescale 1ns/1ps `` - Simulation time unit and precision

`module testbench();` - Testbench top module

`reg clk;` - Clock

`reg rst_n;` - Active-low reset

`reg [3:0] prog_nibble_in;` - Serial 4-bit input

`reg prog_nibble_in_valid;` - input valid

`wire [3:0] prog_nibble_out;` - output nibble

`wire prog_out_valid;` - outpu valid

`wire prog_done;` - Program completion

`wire [11:0] mem_addr;` - Memory address

`wire [31:0] mem_wdata;` - Memory write data

`wire [31:0] mem_rdata;` - Memory read data

`wire mem_wen;` - Memory write enable

`wire mem_cen;` - Memory chip enable

`cpu_top dut(...)` - CPU DUT instantiation

`memory mem_inst(...)` - Memory model instantiation

`always #50 clk = ~clk;` - 100 ns clock generation

`initial begin` - Start stimulus

`clk = 0;` - Initialize clock

`rst_n = 0;` - Assert reset

`repeat(4) @(negedge clk);` - Hold reset

`rst_n = 1;` - Deassert reset

`@(negedge clk);` - Clock alignment

`// Load 32'h12345678 to memory address 12'hF0 - CMD_WRITE`

`prog_nibble_in_valid = 1;` - Start command
`prog_nibble_in = 4'h0;` - Address / opcode / data sequence
`@(negedge clk);` - Continue frame transmission
`prog_nibble_in_valid = 0;` - End command

`repeat(2) @(negedge clk);` - Wait

`// Load 32'h87654321 to memory address 12'hF1 - CMD_WRITE`

`prog_nibble_in_valid = 1;` - Start command
`prog_nibble_in = 4'h1;` - Address / opcode / data sequence
`@(negedge clk);` - Continue frame transmission
`prog_nibble_in_valid = 0;` - End command

`repeat(2) @(negedge clk);` - Wait

`// Add mem(0x0F2) = mem(0x0F1) + mem(0x0F0) - CMD_ADD`

`prog_nibble_in_valid = 1;` - Start ADD command
`prog_nibble_in = 4'h0;` - Address / opcode sequence
`repeat(8) @(negedge clk);` - Dummy data padding
`prog_nibble_in_valid = 0;` - End command

`repeat(2) @(negedge clk);` - Wait

`// Read mem(0xF2) - CMD_READ`

`prog_nibble_in_valid = 1;` - Start READ command
`prog_nibble_in = 4'h2;` - Address / opcode sequence
`repeat(8) @(negedge clk);` - Dummy data padding
`prog_nibble_in_valid = 0;` - End command

`repeat(2) @(negedge clk);` - Wait

`// Load 32'hFEDCBA98 to memory address 12'hF3 - CMD_WRITE`

`prog_nibble_in_valid = 1;` - Start command
`prog_nibble_in = 4'h3;` - Address / opcode / data sequence
`@(negedge clk);` - Continue frame transmission
`prog_nibble_in_valid = 0;` - End command

`repeat(2) @(negedge clk);` - Wait

`// Multiply {mem(0xF5),mem(0xF4)} = mem(0x0F3) × mem(0x0F2) - CMD_MUL`

`prog_nibble_in_valid = 1;` - Start MUL command
`prog_nibble_in = 4'h2;` - Address / opcode sequence
`repeat(8) @(negedge clk);` - Dummy data padding
`prog_nibble_in_valid = 0;` - End command

`repeat(2) @(negedge clk);` - Wait

`// Read mem(0xF4) - CMD_READ`

`prog_nibble_in_valid = 1;` - Start READ command
`prog_nibble_in = 4'h4;` - Address / opcode sequence
`repeat(8) @(negedge clk);` - Dummy data padding
`prog_nibble_in_valid = 0;` - End command

`repeat(2) @(negedge clk);` - Wait

`// Read mem(0xF5) - CMD_READ`

`prog_nibble_in_valid = 1;` - Start READ command
`prog_nibble_in = 4'h5;` - Address / opcode sequence
`repeat(8) @(negedge clk);` - Dummy data padding
`prog_nibble_in_valid = 0;` - End command

`repeat(18) @(negedge clk);` - Allow serializer to finish

`$stop;` - Stop simulation

`end` - End stimulus

`endmodule` - End testbench

Running the testbench:

`run -all`
`csh` - Start C-shell

`source /home/mentor/test.cshrc` - Load Questa environment

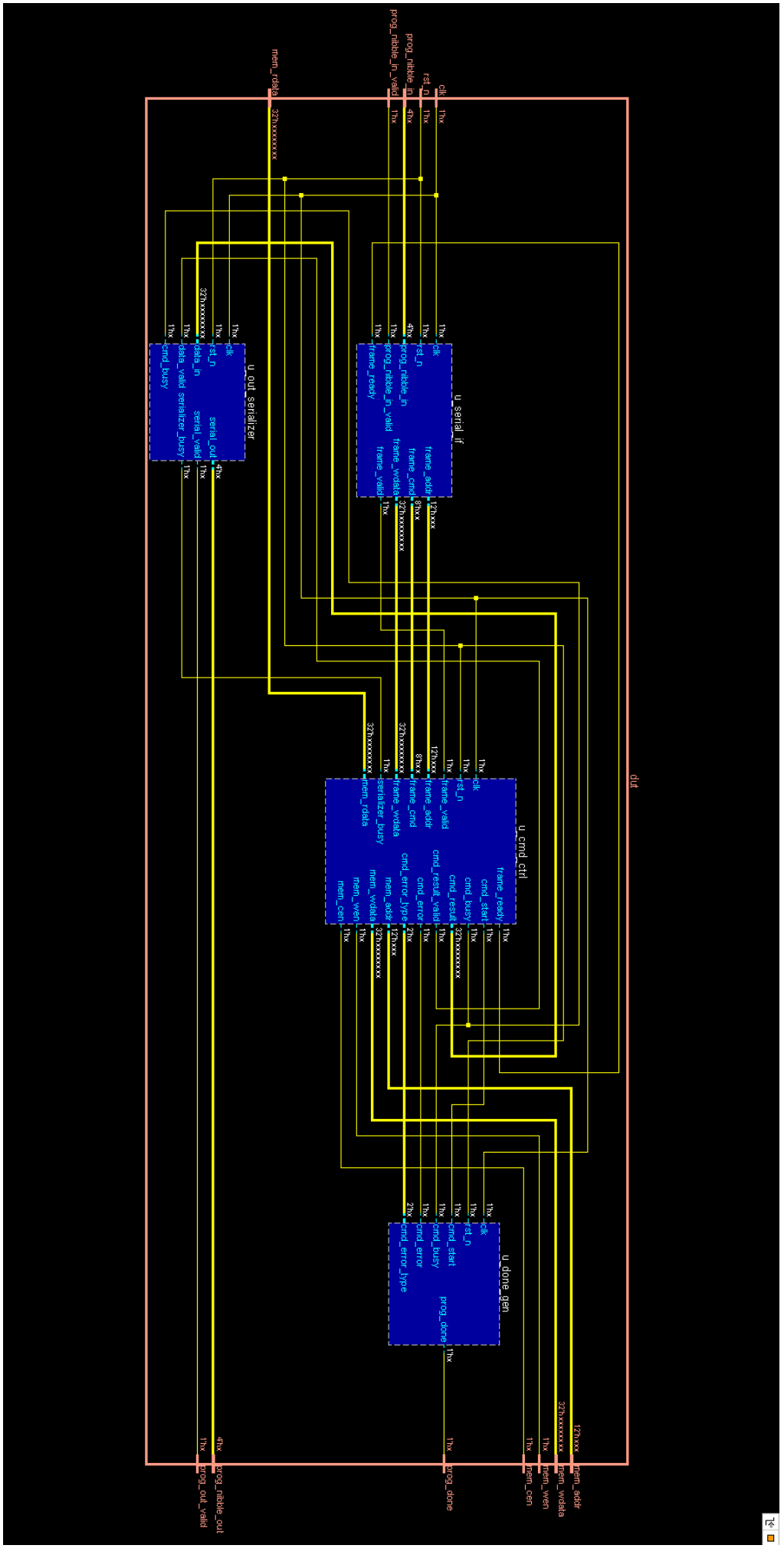
`vlog -64 +acc testbench.v memory.v serial_interface.sv output_serializer.sv done_generator.sv cpu_top.sv cmd_controller.sv` - Compile design with full visibility

`vsim -64 testbench -do "log -r /*"` - Start simulation and log all signals

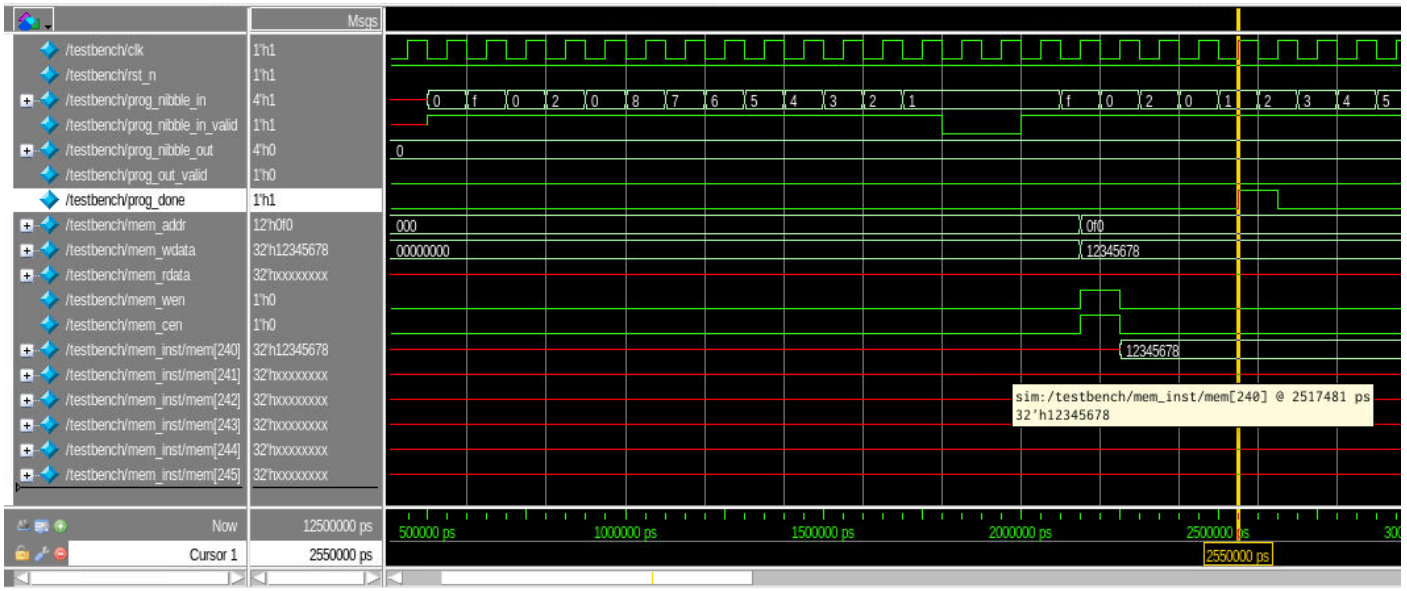
`run -all` - Run simulation

EC22B1004 - P Kaarthick Natesh

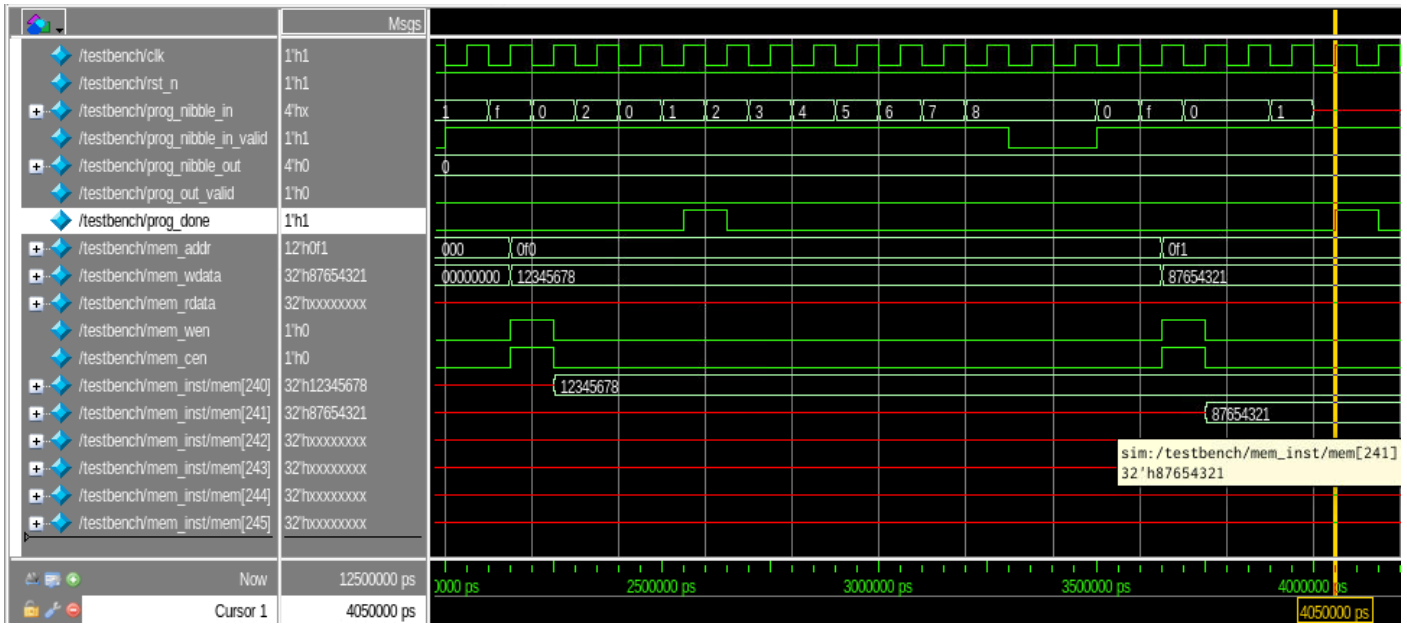
Schematic :



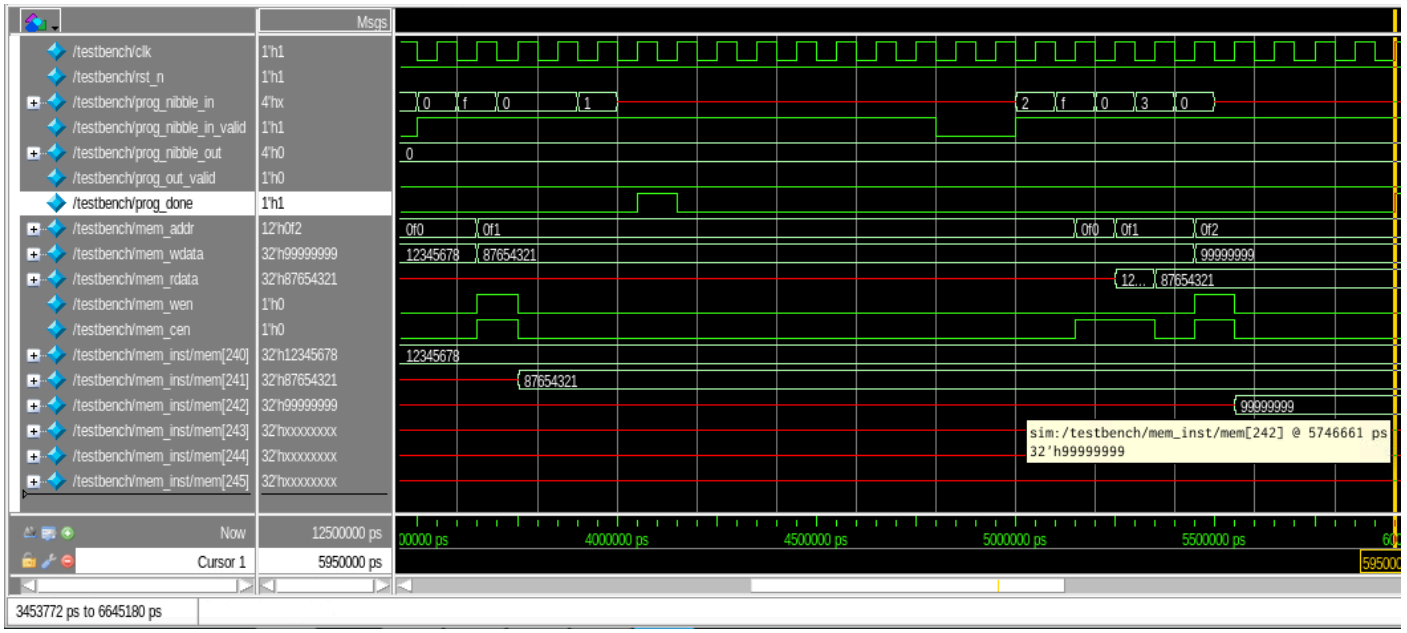
Output Waveforms:



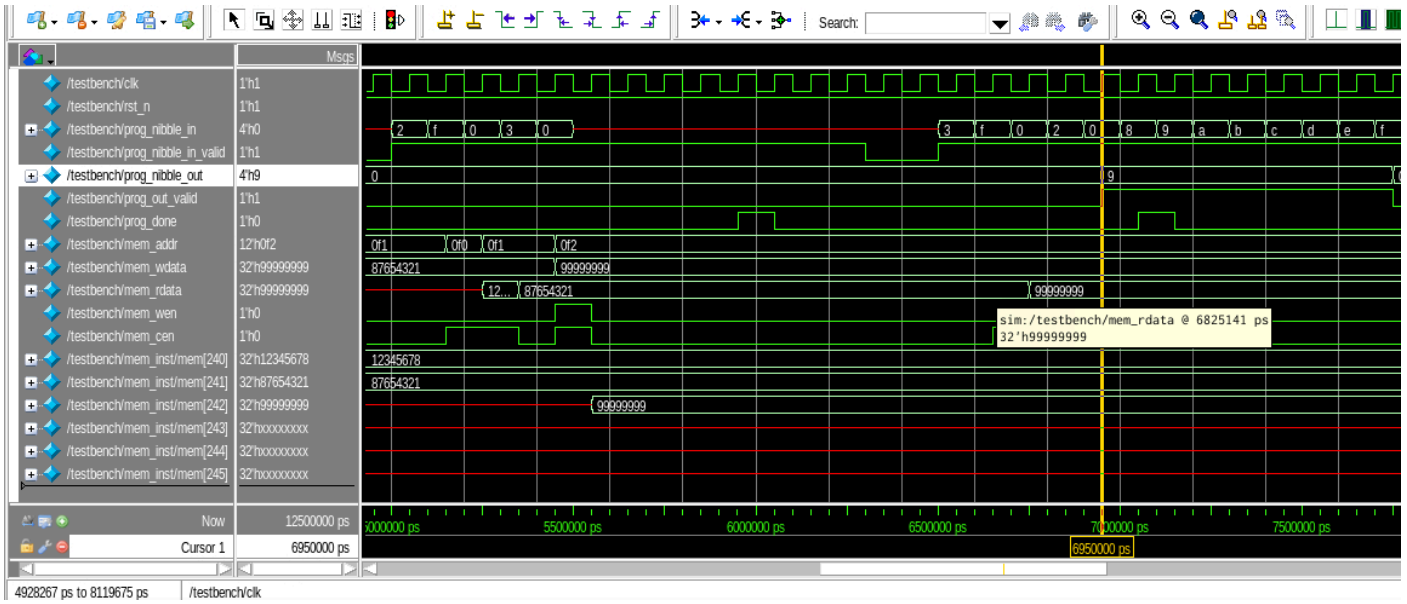
1. **Load 32'h12345678 to memory address 12'hF0 – CMD_WRITE**
    a. `12'h0F0`
    b. `8'h02`
    c. `32'h12345678)`



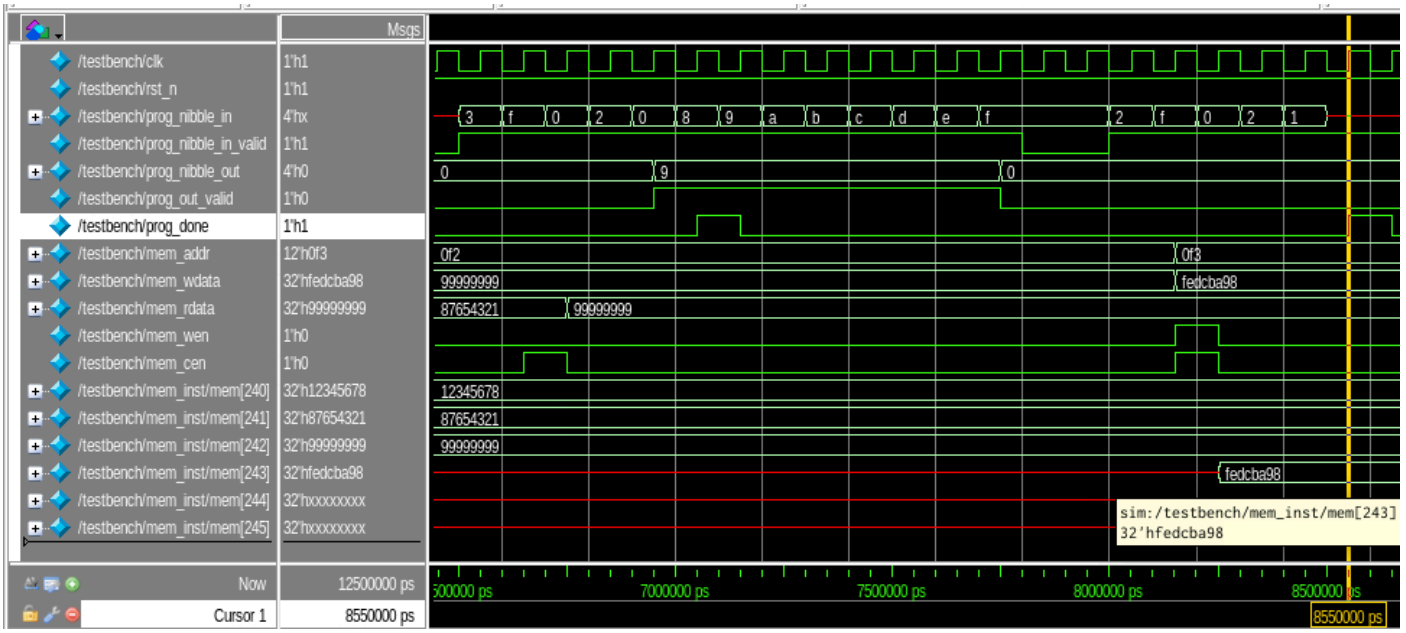
2. **Load 32'h87654321 to memory address 12'hF1 - CMD_WRITE**
    a. `12'h0F1`
    b. `8'h02`
    c. `32'h87654321`



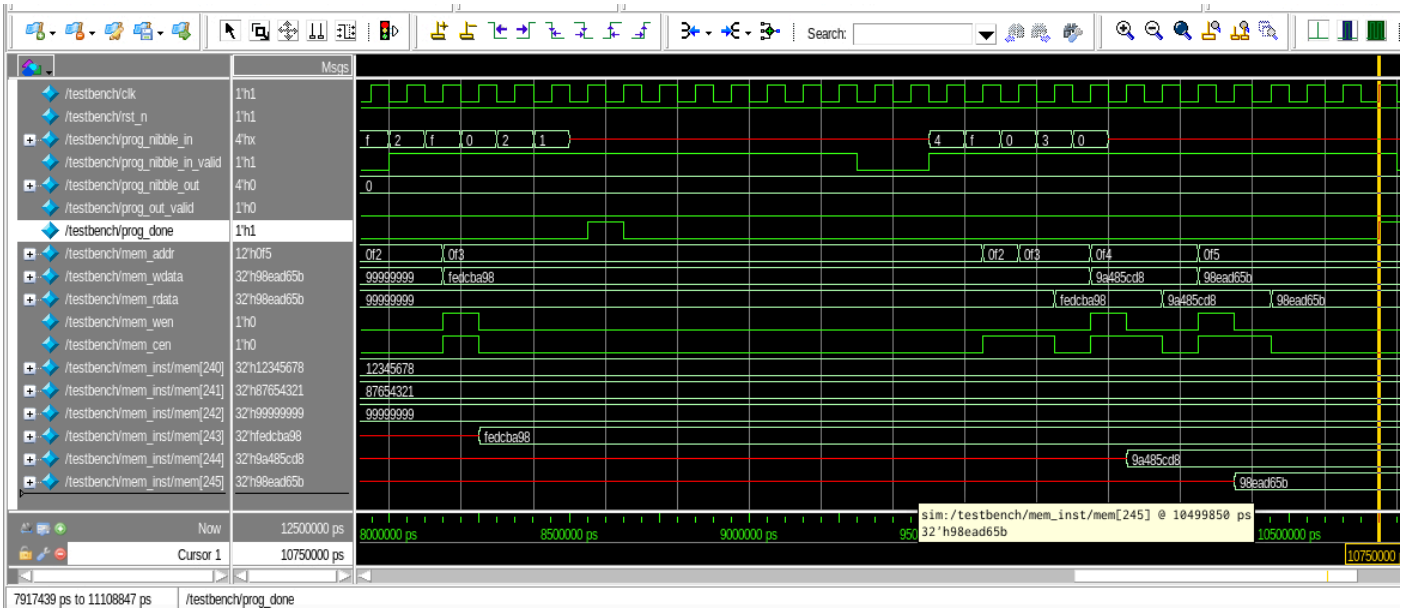
3. **Add mem(0x0F2) = mem(0x0F1) + mem(0x0F0) - CMD_ADD**
    a. 12'h0F0
    b. 8'h10
    c. 32'hX
        i. 32’h12345678 + 32’h87654321 = 32’h99999999



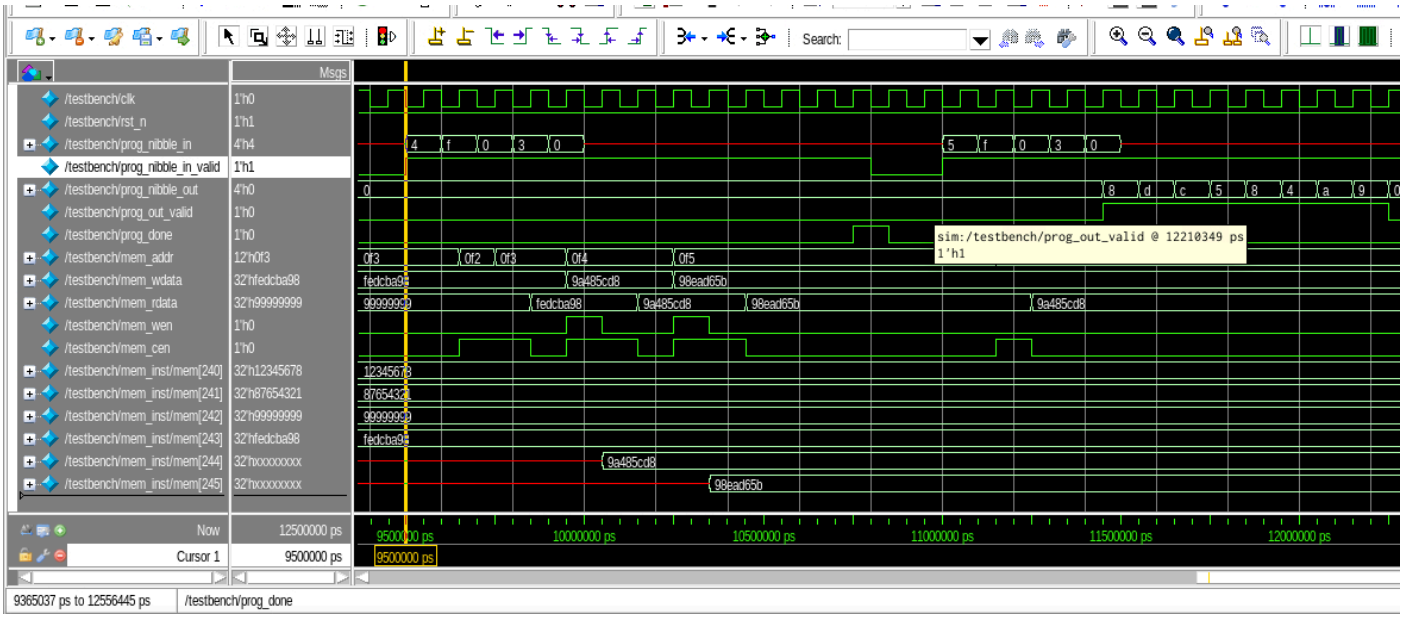
4. **Read mem(0xF2) - CMD_READ**
    a. 12'h0F2
    b. 8'h03
    c. 32'hX



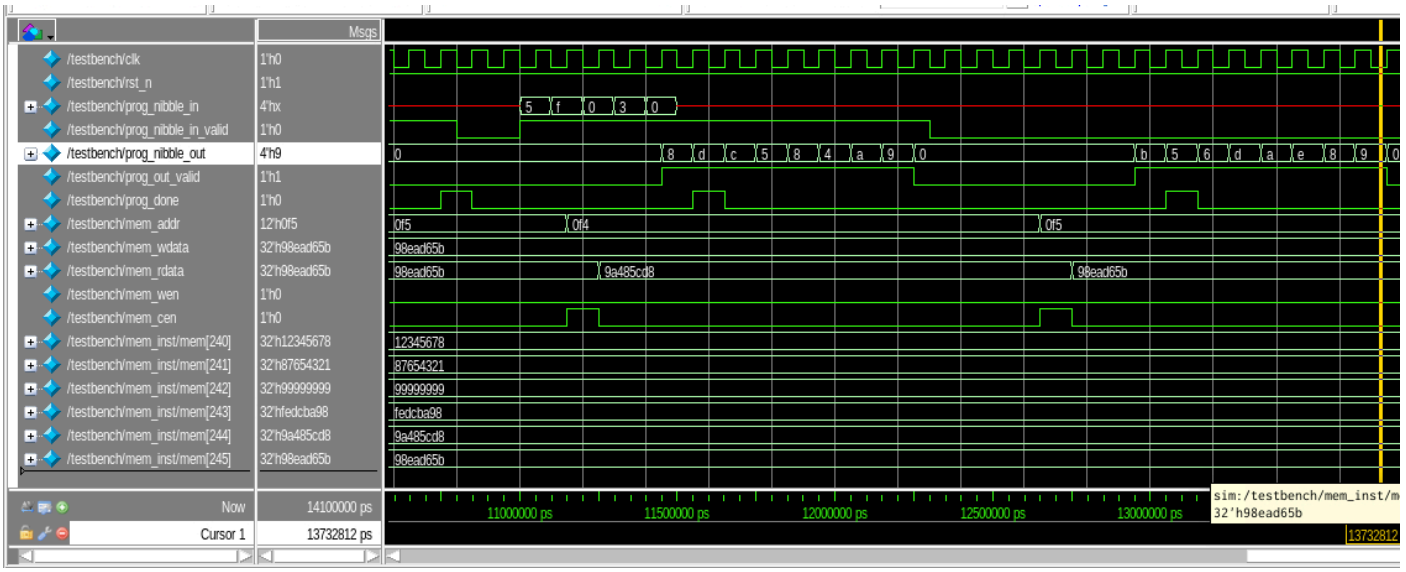
5. **Load 32'hfedcba98 to memory address 12'hF3 - CMD_WRITE**
   a. 12'h0F3
   b. 8'h02
   c. 32'hfedcba98



6. **Multiply {mem(0xF5),mem(0xF4)} = mem(0x0F3) + mem(0x0F2) - CMD_MUL**
   a. 12'h0F3
   b. 8'h12
   c. 32'hX
      i. 32'h99999999 * 32'hfedcba98 = {32'h98ead65b,32'h9a485cd8}
      ii. But from Calculator 32'h99999999 * 32'hfedcba98 = {32'h98ead65a,32'h9a485cd8}



7. Read mem(0xF4) - CMD_READ
    a. 12'h0F4
    b. 8'h03
    c. 32'hX



8. Read mem(0xF5) - CMD_READ
    a. 12'h0F2
    b. 8'h03
    c. 32'hX

Sources:

1. questa_sim_ref.pdf
2. questa_sim_gui.pdf
3. A Verilog HDL Test Bench Primer - https://people.ece.cornell.edu/land/courses/ece5760/Verilog/LatticeTestbenchPrimer.pdf