Universität Siegen
Naturwissenschaftlich-Technische Fakultät
Department Elektrotechnik und Informatik
Lehrstuhl für Digital Integrated Systems

Masterarbeit

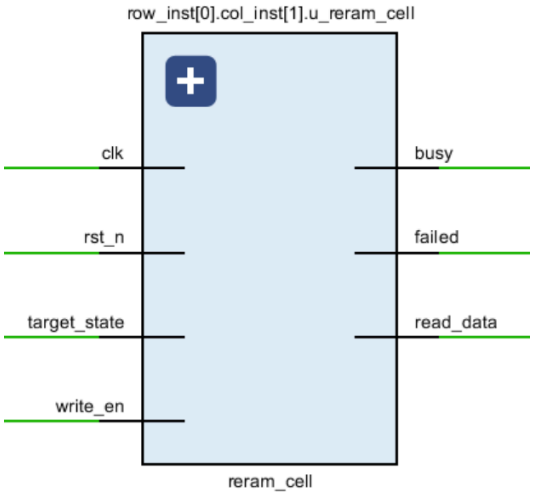
# Evaluation model of a parametrizable ReRAM core in a FPGA architecture

by
Hrishikesh Balkrishna Karande
Matriculation number
1701995

Examiner
Prof. Dr.-Ing. Michael Wahl
2nd Examiner
Prof. Dr.-Ing. Bing Li

Date of delivery
30.10.2025

# 1. reram_cell.sv - ReRAM Cell Behavioral Model




This module models the basic electrical and endurance characteristics of a single ReRAM cell. It's a behavioral model, meaning it describes *what* the cell does, not *how* it's implemented at the transistor level.

## Module Declaration and Parameters

```
`timescale 1ns / 1ps

module reram_cell #( /*Parameters*/
    parameter LRS_STATE = 1'b0,     // Low Resistance State (e.g., programmed '0')
    parameter HRS_STATE = 1'b1,     // High Resistance State (e.g., erased '1') - This is our default state
    parameter SET_DELAY_CYCLES = 10, // Requires 10 clock cycles for SET operation (changing to LRS)
    parameter RESET_DELAY_CYCLES = 10, // Requires 10 clock cycles for RESET operation (changing to HRS)
    parameter ENDURANCE_LIMIT = 1000 // Can work for 1000 write cycles
) ( /*Port Defs*/
    input wire clk,
    input wire rst_n,                // Active low reset
    input wire write_en,             // Asserts to initiate a write operation
    input wire target_state,         // We tell the cell the state that we have to change to (LRS_STATE or HRS_STATE)
    output wire read_data,           // Tells us the current state of the cell
    output wire busy,                // Tells us that the cell is undergoing a write operation
    output wire failed               // Tells us that the cell has reached the endurance limit
);
```

- **timescale 1ns / 1ps**: Defines the simulation time unit (1ns) and precision (1ps).
- **module reram_cell #(...) (...)**: Standard Verilog/SystemVerilog module definition with parameters and ports.
  - **Parameters (#(...))**: These allow you to customize the cell's behavior without changing the code directly.
    - LRS_STATE and HRS_STATE: Define the logical values representing the Low Resistance State (LRS) and High Resistance State (HRS). Here, LRS is '0' and HRS is '1'. HRS is also the default reset state.
    - SET_DELAY_CYCLES / RESET_DELAY_CYCLES: Number of clock cycles required for a write operation (SET or RESET). This models the time it takes for the resistive switching to occur.

        - ENDURANCE_LIMIT: The maximum number of write cycles (SET or RESET) the cell can reliably perform before it "fails."
    - **Ports ((...))**: The inputs and outputs of the module.
        - clk: Clock signal, synchronizes all sequential logic.
        - rst_n: Active-low reset.
        - write_en: When asserted, it initiates a write.
        - target_state: Specifies whether to SET (to LRS) or RESET (to HRS) the cell.
        - read_data: Outputs the current stored state of the ReRAM cell.
        - busy: Indicates if a write operation is in progress.
        - failed: Indicates if the cell has exceeded its ENDURANCE_LIMIT.

## Internal Signals

```
    /*Internal Signals*/
    //Internal state of the ReRAM cell (LRS_STATE or HRS_STATE)
    reg current_state;

    //Assignment from internal state to output, This is combinational assignment
    assign read_data = current_state;

    //Counter for SET/RESET delays
    reg [31:0] delay_counter;

    //State machine for write operations (IDLE, SETTING, RESETTING)
    localparam IDLE = 2'b00;
    localparam SETTING = 2'b01;
    localparam RESETTING = 2'b10;
    reg [1:0] write_fsm_state;

    //Endurance Counter
    reg [31:0] endurance_counter;
    reg cell_failed_flag;

    assign busy = (write_fsm_state != IDLE);
    assign failed = cell_failed_flag;
```

- **reg current_state;**: A register holding the actual data/state of the ReRAM cell. This is the "memory" element.
- **assign read_data = current_state;**: A continuous assignment. The read_data output simply reflects the current_state combinatorially (immediately).
- **reg [31:0] delay_counter;**: A counter to track the progress of the SET/RESET delay cycles. 32 bits is a generous size for typical delays.
- **localparam IDLE = 2'b00; ...**: Defines the states for the write operation's Finite State Machine (FSM). localparam means these are local to the module and cannot be overridden by external parameters.
    - IDLE: The cell is not performing a write operation.
    - SETTING: The cell is currently undergoing a SET operation.
    - RESETTING: The cell is currently undergoing a RESET operation.
- **reg [1:0] write_fsm_state;**: The register that holds the current state of the write FSM. It's 2 bits because there are 3 states, requiring at least 2 bits (2^2 = 4 possible states).
- **reg [31:0] endurance_counter;**: Counts the number of successful write operations.

- **reg cell_failed_flag;**: A flag that becomes '1' once the endurance_counter reaches ENDURANCE_LIMIT.
- **assign busy = (write_fsm_state != IDLE);**: Continuous assignment. The busy output is '1' whenever the FSM is not in the IDLE state (i.e., it's SETTING or RESETTING).
- **assign failed = cell_failed_flag;**: Continuous assignment. The failed output simply reflects the cell_failed_flag.

## Behavioral Modeling (The always Block)

```
    /*Behavior Modelling*/
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin // Asynchronous Active-Low Reset
            current_state <= HRS_STATE;
            delay_counter <= 0;
            write_fsm_state <= IDLE;
            endurance_counter <= 0;
            cell_failed_flag <= 1'b0;
            $display("Time %0t: Cell RESET! Initial State: %b, Endurance: %0d", $time, HRS_STATE, 0);
        end else begin
            // Check for cell failure due to endurance. This check happens every cycle.
            // If already failed, remain failed. If not failed and limit reached, set flag.
            if (!cell_failed_flag && endurance_counter >= ENDURANCE_LIMIT) begin
                cell_failed_flag <= 1'b1;
                $display("Time %0t: Cell FAILED! Endurance Limit (%0d) reached. Current endurance:
%0d", $time, ENDURANCE_LIMIT, endurance_counter);
            end

            // Main Write FSM
            case (write_fsm_state)
                IDLE: begin
                    // Only start a write if write_en is asserted AND cell has not failed
                    if (write_en && !cell_failed_flag) begin
                        if (target_state == LRS_STATE) begin
                            write_fsm_state <= SETTING;
                            delay_counter <= 0;
                            $display("Time %0t: Cell (IDLE->SETTING) for target %b. Current: %b",
$time, target_state, current_state);
                        end else if (target_state == HRS_STATE) begin
                            write_fsm_state <= RESETTING;
                            delay_counter <= 0;
                            $display("Time %0t: Cell (IDLE->RESETTING) for target %b. Current: %b",
$time, target_state, current_state);
                        end
                    end else if (write_en && cell_failed_flag) begin
                        $display("Time %0t: Cell (IDLE) Write attempt to FAILED cell. Current: %b,
Endurance: %0d", $time, current_state, endurance_counter);
                        // Cell remains in IDLE, state does not change, endurance_counter does not
increment.
                    end
                end // end IDLE

                SETTING: begin
                    if (delay_counter < SET_DELAY_CYCLES - 1) begin // Counting up to DELAY-1, so DELAY
cycles in total
```

```
                    delay_counter <= delay_counter + 1;
                    $display("Time %0t: Cell (SETTING) Delay: %0d/%0d. Current State: %b", $time,
delay_counter, SET_DELAY_CYCLES, current_state);
                end else begin // Delay is complete
                    current_state <= LRS_STATE; // State change happens here
                    endurance_counter <= endurance_counter + 1; // Increment on successful write
operation
                    write_fsm_state <= IDLE; // Changing state back to IDLE
                    $display("Time %0t: Cell (SETTING->IDLE) COMPLETE. New State: %b, Endurance:
%0d", $time, LRS_STATE, endurance_counter + 1);
                end
            end // end SETTING

            RESETTING: begin
                if (delay_counter < RESET_DELAY_CYCLES - 1) begin // Counting up to DELAY-1
                    delay_counter <= delay_counter + 1;
                    $display("Time %0t: Cell (RESETTING) Delay: %0d/%0d. Current State: %b", $time,
delay_counter, RESET_DELAY_CYCLES, current_state);
                end else begin // Delay is complete
                    current_state <= HRS_STATE; // State change happens here
                    endurance_counter <= endurance_counter + 1; // Increment on successful write
operation
                    write_fsm_state <= IDLE; // Changing state back to IDLE
                    $display("Time %0t: Cell (RESETTING->IDLE) COMPLETE. New State: %b, Endurance:
%0d", $time, HRS_STATE, endurance_counter + 1);
                end
            end // end RESETTING

            default: begin // This state should never be reached, if ever reached then take the
machine to IDLE
                write_fsm_state <= IDLE;
            end
        endcase
    end // end else (not reset)
end // end always
```

- **always @(posedge clk or negedge rst_n) begin ... end**: This is a sequential block, meaning its contents are executed on the positive edge of clk or the negative edge of rst_n. All assignments within this block use non-blocking assignments (<=), which is crucial for synchronous logic.
- **Reset Logic (if (!rst_n))**:
  - When rst_n is low (active-low reset), all internal registers are initialized to their default values.
  - current_state is set to HRS_STATE.
  - Counters and FSM state are cleared.
  - cell_failed_flag is reset to 0.
  - A $display message confirms the reset.
- **Endurance Check (else if (!cell_failed_flag && endurance_counter >= ENDURANCE_LIMIT))**:
  - This logic runs on every positive clock edge (when not in reset).
  - It continuously checks if the endurance_counter has reached or exceeded the ENDURANCE_LIMIT *and* if the cell_failed_flag isn't already set.
  - If the limit is reached, cell_failed_flag is set to 1'b1, and a $display message announces the cell failure.
  - Crucially, once cell_failed_flag is set, it stays set (!cell_failed_flag becomes false, so this if condition won't trigger again

for the same cell).

- **Main Write FSM (case (write_fsm_state))**: This case statement defines the behavior of the cell based on its current write_fsm_state.
  - **IDLE State:**
    - **Condition to start write:** Checks if (write_en && !cell_failed_flag). A write can only start if write_en is asserted *and* the cell has *not* failed.
    - **Determine target:**
      - If target_state is LRS_STATE, the FSM transitions to SETTING. delay_counter is reset.
      - If target_state is HRS_STATE, the FSM transitions to RESETTING. delay_counter is reset.
    - **Attempt to write failed cell:** else if (write_en && cell_failed_flag): If write_en is asserted but the cell has already failed, a message is displayed, but the cell remains in IDLE. No state change occurs, and endurance_counter is *not* incremented, modeling a failed write attempt.
  - **SETTING State:**
    - **Delay countdown:** if (delay_counter < SET_DELAY_CYCLES - 1): The FSM stays in SETTING for SET_DELAY_CYCLES. The delay_counter increments each cycle. The comparison SET_DELAY_CYCLES - 1 ensures that it counts from 0 up to SET_DELAY_CYCLES - 1, covering exactly SET_DELAY_CYCLES clock cycles (e.g., for 10 cycles, it counts 0, 1, ..., 9).
    - **Completion:** else: When the delay is complete (delay_counter is SET_DELAY_CYCLES - 1 and the next clock edge arrives):
      - current_state <= LRS_STATE;: The actual ReRAM cell state is updated to LRS.
      - endurance_counter <= endurance_counter + 1;: The endurance counter is incremented, as a successful write occurred.
      - write_fsm_state <= IDLE;: The FSM returns to the IDLE state.
      - A $display message confirms the completion.
  - **RESETTING State:**
    - Similar to SETTING, but the delay is governed by RESET_DELAY_CYCLES.
    - Upon completion, current_state <= HRS_STATE;, endurance_counter increments, and the FSM returns to IDLE.
  - **default State:** A safety net. If the FSM somehow enters an undefined state, it immediately transitions back to IDLE to prevent unpredictable behavior.

# 2. reram_cell_tb.sv - ReRAM Cell Testbench

This module is solely for verifying the behavior of reram_cell. It provides inputs, generates a clock, and checks outputs. It does *not* get synthesized into hardware.

## Module Declaration and Testbench Parameters

```
`timescale 1ns / 1ps

module reram_cell_tb;

    // Parameters for the testbench (should match or be compatible with DUT)
    localparam LRS_STATE_TB = 1'b0;
    localparam HRS_STATE_TB = 1'b1;
    localparam SET_DELAY_CYCLES_TB = 10;
    localparam RESET_DELAY_CYCLES_TB = 10;
    localparam ENDURANCE_LIMIT_TB = 1000;
    localparam CLOCK_PERIOD = 10; // 10ns clock period (100 MHz clock)
```

- **module reram_cell_tb;**: Standard module declaration. Testbenches typically don't have ports because they are the top-level entity in simulation.
- **localparam ..._TB**: These parameters are defined locally in the testbench. It's good practice to mirror the DUT's parameters here, sometimes with different names (e.g., _TB suffix) to avoid confusion. This allows the testbench to easily adapt if the DUT's parameters change.
- **CLOCK_PERIOD**: Defines the clock period for the testbench's clock generator.

## Testbench Signals and DUT Instantiation

```
    // Testbench Signals (wires and regs)
    reg clk;
    reg rst_n;
    reg write_en;
    reg target_state;
    wire read_data;
    wire busy;
    wire failed;
    reg [31:0] i; // Loop counter, declared at module level for XSim compatibility

    // Instantiate the Device Under Test (DUT)
    reram_cell #(
        .LRS_STATE(LRS_STATE_TB),
        .HRS_STATE(HRS_STATE_TB),
        .SET_DELAY_CYCLES(SET_DELAY_CYCLES_TB),
        .RESET_DELAY_CYCLES(RESET_DELAY_CYCLES_TB),
        .ENDURANCE_LIMIT(ENDURANCE_LIMIT_TB)
    ) dut (
        .clk(clk),
        .rst_n(rst_n),
        .write_en(write_en),
        .target_state(target_state),
        .read_data(read_data),
```

```
        .busy(busy),
        .failed(failed)
    );
```

- **reg ...;**: Signals that will drive inputs to the DUT (clk, rst_n, write_en, target_state) must be declared as reg.
- **wire ...;**: Signals that receive outputs from the DUT (read_data, busy, failed) must be declared as wire.
- **reg [31:0] i;**: This is the loop counter variable. It was moved to the module scope as reg to resolve the Vivado XSim syntax errors, ensuring compatibility.
- **reram_cell #(...) dut (...)**: Instantiates your reram_cell module.
  - #(...): Parameter overrides. The testbench's _TB parameters are passed to the DUT.
  - dut: The instance name (arbitrary).
  - (...): Port connections. Signals declared in the testbench are connected to the corresponding ports of the dut.

## Clock Generation

```
    // Clock Generation
    initial begin
        clk = 0;
        forever #(CLOCK_PERIOD / 2) clk = ~clk; // Toggle clock every half period
    end
```

- **initial begin ... end**: This block executes only once at the beginning of the simulation.
- **clk = 0;**: Initializes the clock to low.
- **forever #(CLOCK_PERIOD / 2) clk = ~clk;**: This creates a continuous clock signal.
  - #(CLOCK_PERIOD / 2): Delays for half a clock period.
  - clk = ~clk;: Toggles the clk signal. This results in a square wave with the specified CLOCK_PERIOD.

## Test Scenario (initial block)

This is the heart of the testbench, defining the sequence of operations to test the reram_cell.

```
    initial begin
        $display("------------------------------------------------------");
        $display("           ReRAM Cell Testbench Started             ");
        $display("------------------------------------------------------");

        // Initialize signals
        rst_n = 0; // Assert reset
        write_en = 0;
        target_state = HRS_STATE_TB;

        // Apply reset for a few clock cycles
        #(CLOCK_PERIOD * 5); // Hold reset for 5 clock cycles (50ns in this case)
        rst_n = 1; // De-assert reset
        $display("Time %0t: Reset Released. Initializing test sequence.", $time);

        // --- Test Case 1: Initial State Check ---
        #(CLOCK_PERIOD); // Wait one cycle after reset for signals to settle
        // ... (assertions/checks) ...

        // --- Test Case 2: SET Operation (HRS -> LRS) ---
        $display("Time %0t: Starting SET operation (HRS -> LRS).", $time);
        write_en = 1;
        target_state = LRS_STATE_TB;
```

```
        #(CLOCK_PERIOD); // Assert write_en for one full clock cycle
        write_en = 0; // De-assert write_en; DUT should now be busy setting

        // Wait for SET operation to complete
        wait (dut.write_fsm_state == dut.IDLE); // Waits until DUT's FSM returns to IDLE
        #(CLOCK_PERIOD); // Give an extra cycle for read_data to update after FSM transitions
        // ... (assertions/checks) ...

        // --- Test Case 3: RESET Operation (LRS -> HRS) ---
        $display("Time %0t: Starting RESET operation (LRS -> HRS).", $time);
        write_en = 1;
        target_state = HRS_STATE_TB;
        #(CLOCK_PERIOD);
        write_en = 0;

        // Wait for RESET operation to complete
        wait (dut.write_fsm_state == dut.IDLE);
        #(CLOCK_PERIOD);
        // ... (assertions/checks) ...

        // --- Test Case 4: Repeated Writes to hit Endurance Limit ---
        $display("Time %0t: Starting repeated write operations to test endurance limit (%0d).",
                 $time, ENDURANCE_LIMIT_TB);
        for (i = 0; i < ENDURANCE_LIMIT_TB + 5; i++) begin // Iterate slightly more than limit
            if (!failed) begin // Only attempt write if cell hasn't failed yet
                write_en = 1;
                target_state = (read_data == HRS_STATE_TB) ? LRS_STATE_TB : HRS_STATE_TB; // Alternate
state
                #(CLOCK_PERIOD);
                write_en = 0;

                wait (dut.write_fsm_state == dut.IDLE);
                #(CLOCK_PERIOD);

                $display("Time %0t: Write Cycle %0d. Current State: %b, Busy: %b, Failed: %b",
                         $time, i + 1, read_data, busy, failed);

                // Basic check after each write
                if (!failed && (read_data !== ((target_state == LRS_STATE_TB) ? LRS_STATE_TB :
HRS_STATE_TB))) begin
                    $error("Write cycle %0d FAILED! State mismatch.", i + 1);
                end
            end else begin
                $display("Time %0t: Cell FAILED at cycle %0d. Remaining cycles will attempt writes to a
failed cell.", $time, i);
                break; // Exit loop if cell fails
            end
        end

        // --- Test Case 5: Attempt Write to Failed Cell ---
        $display("Time %0t: Attempting write to a FAILED cell.", $time);
```

```
        if (failed) begin
            write_en = 1;
            target_state = (read_data == HRS_STATE_TB) ? LRS_STATE_TB : HRS_STATE_TB;
            #(CLOCK_PERIOD * (SET_DELAY_CYCLES_TB + 5)); // Provide ample time to see if busy
            write_en = 0;
            $display("Time %0t: Write attempt on failed cell completed. Busy: %b, Failed: %b, Read
Data: %b",
                     $time, busy, failed, read_data);
            if (busy !== 0) begin
                $error("Failed cell should not be busy after write attempt.");
            end
        end else begin
            $warning("Cell did not fail during endurance test, skipping 'write to failed cell' test.");
        end

        $display("-----------------------------------------------------");
        $display("           ReRAM Cell Testbench Finished           ");
        $display("-----------------------------------------------------");
        $finish; // End simulation
    end
```

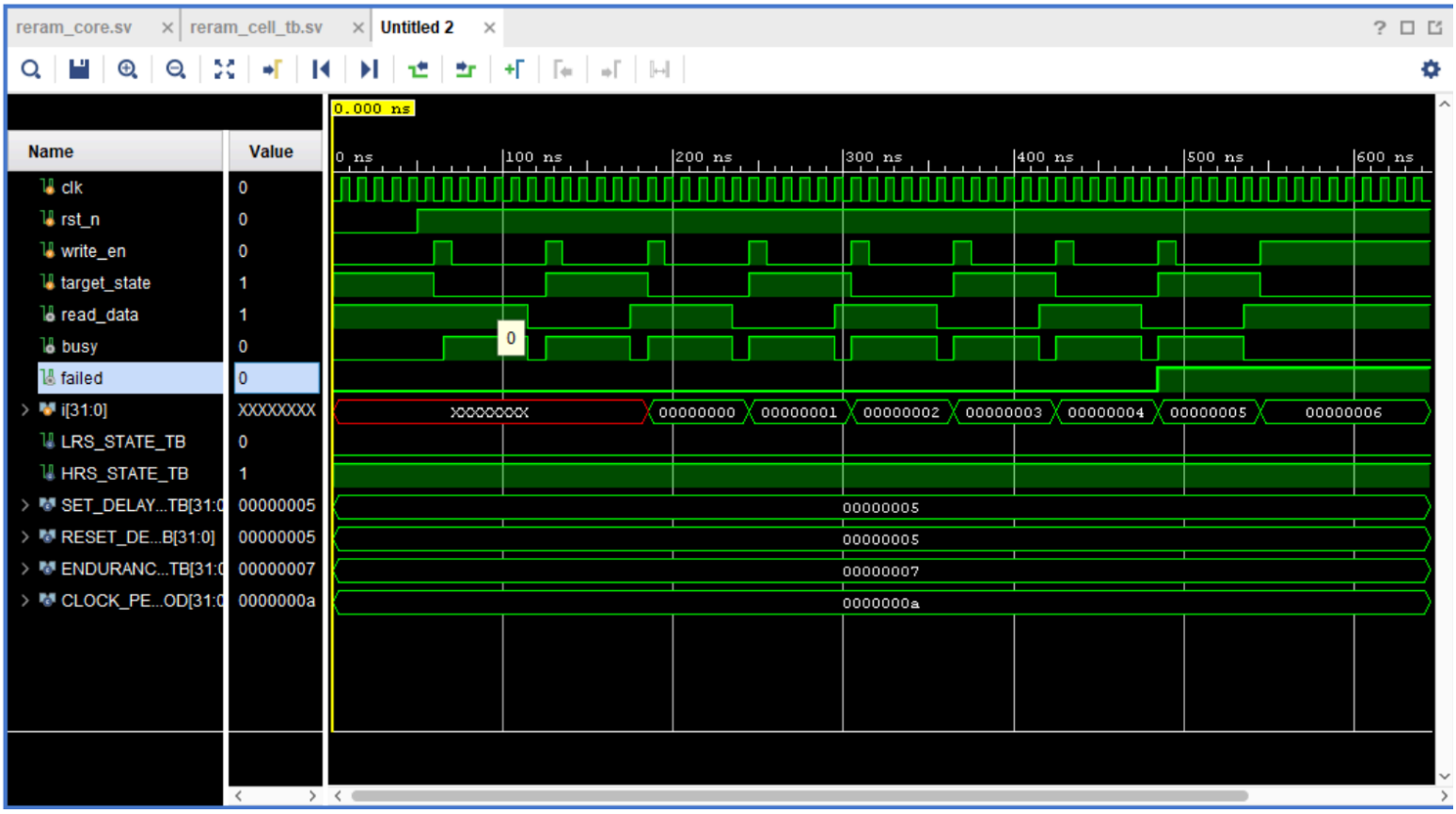
- **Sequential Execution:** The initial block executes its statements sequentially.
- **Signal Initialization:** Sets all inputs to known, safe values at the beginning.
- **Reset Sequence:** It brings rst_n low for 5 clock cycles, then raises it, modeling a proper reset.
- **#(delay):** This is a Verilog time delay. #(CLOCK_PERIOD) waits for one clock cycle's duration. This is important for synchronous logic where inputs need to be stable before the clock edge.
- **write_en pulse:** For write operations, write_en is asserted for exactly one clock cycle (write_en = 1; #(CLOCK_PERIOD); write_en = 0;). The DUT's FSM should then take over and manage the delay.
- **wait (condition);**: This statement pauses the execution of the initial block until the specified condition becomes true. This is crucial for synchronizing the testbench with the DUT's internal state, ensuring the testbench doesn't try to send a new command while the DUT is still busy.
- **$display(...)**: Prints messages to the simulation console, showing the progress of the test and key values.
- **$error(...)**: Prints an error message and increments an internal error count in the simulator. Good for automated checking.
- **$warning(...)**: Prints a warning message.
- **Endurance Test Loop:** The for loop repeatedly issues write commands, alternating the target_state to stress the cell's endurance. It checks the failed flag to ensure no new writes are attempted on a failed cell.
- **Post-Failure Test:** After the endurance limit is theoretically hit, it attempts one more write to a "failed" cell to confirm that the busy signal remains low and the state doesn't change, as per the DUT's specification.
- **$finish;**: Terminates the simulation.

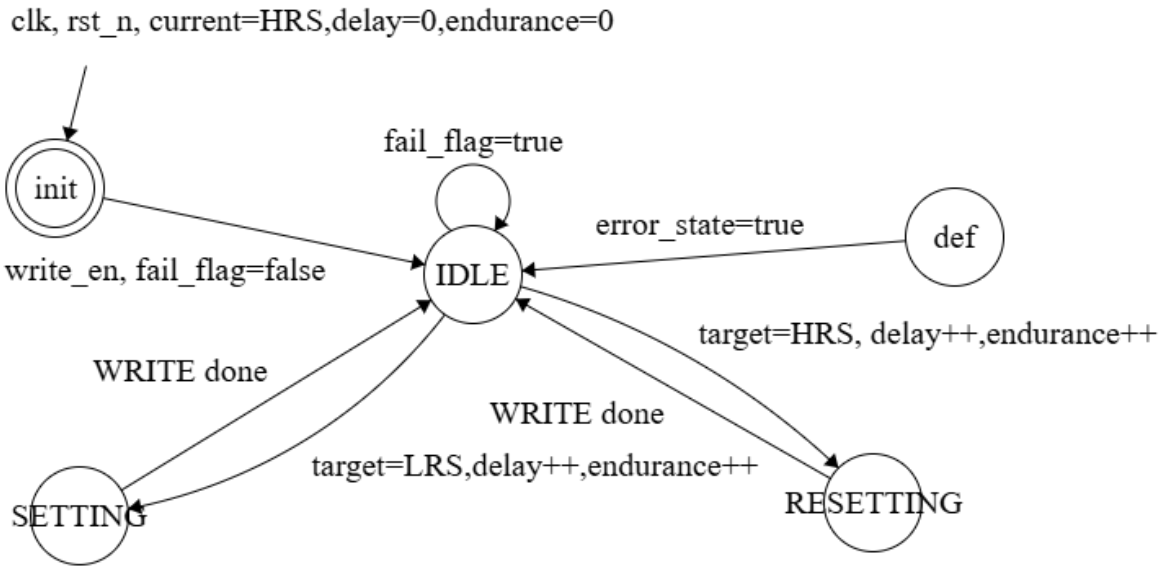
```
launch_simulation
INFO: [Vivado 12-5682] Launching behavioral simulation in
'C:/Users/hrish/Downloads/ReRAM_Thesis/Project Folder/Sim/Sim.sim/sim_1/behav/xsim'
INFO: [SIM-utils-51] Simulation object is 'sim_1'
INFO: [SIM-utils-54] Inspecting design source files for 'reram_cell_tb' in fileset 'sim_1'...
INFO: [USF-XSim-97] Finding global include files...
INFO: [USF-XSim-98] Fetching design files from 'sim_1'...
INFO: [USF-XSim-2] XSim::Compile design
INFO: [USF-XSim-61] Executing 'COMPILE and ANALYZE' step in
```

```
'C:/Users/hrish/Downloads/ReRAM_Thesis/Project Folder/Sim/Sim.sim/sim_1/behav/xsim'
"xvlog --incr --relax -L xil_defaultlib -prj reram_cell_tb_vlog.prj"
INFO: [VRFC 10-2263] Analyzing SystemVerilog file "C:/Users/hrish/Downloads/ReRAM_Thesis/Project
Folder/Sim/Sim.srcs/sources_1/new/reram_cell.sv" into library xil_defaultlib
INFO: [VRFC 10-311] analyzing module reram_cell
INFO: [VRFC 10-2263] Analyzing SystemVerilog file "C:/Users/hrish/Downloads/ReRAM_Thesis/Project
Folder/Sim/Sim.srcs/sim_1/new/reram_cell_tb.sv" into library xil_defaultlib
INFO: [VRFC 10-311] analyzing module reram_cell_tb
INFO: [USF-XSim-69] 'compile' step finished in '1' seconds
INFO: [USF-XSim-3] XSim::Elaborate design
INFO: [USF-XSim-61] Executing 'ELABORATE' step in 'C:/Users/hrish/Downloads/ReRAM_Thesis/Project
Folder/Sim/Sim.sim/sim_1/behav/xsim'
Vivado Simulator 2018.1
Copyright 1986-1999, 2001-2017 Xilinx, Inc. All Rights Reserved.
Running: C:/Xilinx/Vivado/2018.1/bin/unwrapped/win64.o/xelab.exe -wto 45fd118e296e4cf9908420759dbb98b6
--incr --debug typical --relax --mt 2 -L xilinx_vip -L xil_defaultlib -L unisims_ver -L unimacro_ver -L
secureip --snapshot reram_cell_tb_behav xil_defaultlib.reram_cell_tb xil_defaultlib.glbl -log
elaborate.log
Using 2 slave threads.
Starting static elaboration
Completed static elaboration
Starting simulation data flow analysis
Completed simulation data flow analysis
Time Resolution for simulation is 1ps
Compiling module xil_defaultlib.reram_cell(SET_DELAY_CYCLES=5,RE...
Compiling module xil_defaultlib.reram_cell_tb
Compiling module xil_defaultlib.glbl
Built simulation snapshot reram_cell_tb_behav
INFO: [USF-XSim-69] 'elaborate' step finished in '2' seconds
INFO: [USF-XSim-4] XSim::Simulate design
INFO: [USF-XSim-61] Executing 'SIMULATE' step in 'C:/Users/hrish/Downloads/ReRAM_Thesis/Project
Folder/Sim/Sim.sim/sim_1/behav/xsim'
INFO: [USF-XSim-98] *** Running xsim
   with args "reram_cell_tb_behav -key {Behavioral:sim_1:Functional:reram_cell_tb} -tclbatch
{reram_cell_tb.tcl} -log {simulate.log}"
INFO: [USF-XSim-8] Loading simulator feature
Vivado Simulator 2018.1
Time resolution is 1 ps
source reram_cell_tb.tcl
# set curr_wave [current_wave_config]
# if { [string length $curr_wave] == 0 } {
#   if { [llength [get_objects]] > 0} {
#     add_wave /
#     set_property needs_save false [current_wave_config]
#   } else {
#      send_msg_id Add_Wave-1 WARNING "No top level signals found. Simulator will start without a wave
window. If you want to open a wave window go to 'File->New Waveform Configuration' or type
'create_wave_config' in the TCL console."
#   }
# }
# run 1000ms
```

```
--------------------------------------------------
           ReRAM Cell Testbench Started
--------------------------------------------------
Time 0: Cell RESET! Initial State: 1, Endurance: 0
Time 5000: Cell RESET! Initial State: 1, Endurance: 0
Time 15000: Cell RESET! Initial State: 1, Endurance: 0
Time 25000: Cell RESET! Initial State: 1, Endurance: 0
Time 35000: Cell RESET! Initial State: 1, Endurance: 0
Time 45000: Cell RESET! Initial State: 1, Endurance: 0
Time 50000: Reset Released. Initializing test sequence.
Time 60000: Initial state check. Expected: 1, Actual: 1. Failed: 0
Time 60000: Starting SET operation (HRS -> LRS).
Time 65000: Cell (IDLE->SETTING) for target 0. Current: 1
Time 75000: Cell (SETTING) Delay: 0/5. Current State: 1
Time 85000: Cell (SETTING) Delay: 1/5. Current State: 1
Time 95000: Cell (SETTING) Delay: 2/5. Current State: 1
Time 105000: Cell (SETTING) Delay: 3/5. Current State: 1
Time 115000: Cell (SETTING->IDLE) COMPLETE. New State: 0, Endurance: 1
Time 125000: SET operation completed. Expected: 0, Actual: 0. Busy: 0, Failed: 0
Time 125000: Starting RESET operation (LRS -> HRS).
Time 125000: Cell (IDLE->RESETTING) for target 1. Current: 0
Time 135000: Cell (RESETTING) Delay: 0/5. Current State: 0
Time 145000: Cell (RESETTING) Delay: 1/5. Current State: 0
Time 155000: Cell (RESETTING) Delay: 2/5. Current State: 0
Time 165000: Cell (RESETTING) Delay: 3/5. Current State: 0
Time 175000: Cell (RESETTING->IDLE) COMPLETE. New State: 1, Endurance: 2
Time 185000: RESET operation completed. Expected: 1, Actual: 1. Busy: 0, Failed: 0
Time 185000: Starting repeated write operations to test endurance limit (7).
Time 185000: Cell (IDLE->SETTING) for target 0. Current: 1
Time 195000: Cell (SETTING) Delay: 0/5. Current State: 1
Time 205000: Cell (SETTING) Delay: 1/5. Current State: 1
Time 215000: Cell (SETTING) Delay: 2/5. Current State: 1
Time 225000: Cell (SETTING) Delay: 3/5. Current State: 1
Time 235000: Cell (SETTING->IDLE) COMPLETE. New State: 0, Endurance: 3
Time 245000: Write Cycle 1. Current State: 0, Busy: 0, Failed: 0
Time 245000: Cell (IDLE->RESETTING) for target 1. Current: 0
Time 255000: Cell (RESETTING) Delay: 0/5. Current State: 0
Time 265000: Cell (RESETTING) Delay: 1/5. Current State: 0
Time 275000: Cell (RESETTING) Delay: 2/5. Current State: 0
Time 285000: Cell (RESETTING) Delay: 3/5. Current State: 0
Time 295000: Cell (RESETTING->IDLE) COMPLETE. New State: 1, Endurance: 4
Time 305000: Write Cycle 2. Current State: 1, Busy: 0, Failed: 0
Time 305000: Cell (IDLE->SETTING) for target 0. Current: 1
Time 315000: Cell (SETTING) Delay: 0/5. Current State: 1
Time 325000: Cell (SETTING) Delay: 1/5. Current State: 1
Time 335000: Cell (SETTING) Delay: 2/5. Current State: 1
Time 345000: Cell (SETTING) Delay: 3/5. Current State: 1
Time 355000: Cell (SETTING->IDLE) COMPLETE. New State: 0, Endurance: 5
Time 365000: Write Cycle 3. Current State: 0, Busy: 0, Failed: 0
Time 365000: Cell (IDLE->RESETTING) for target 1. Current: 0
Time 375000: Cell (RESETTING) Delay: 0/5. Current State: 0
```

```
Time 385000: Cell (RESETTING) Delay: 1/5. Current State: 0
Time 395000: Cell (RESETTING) Delay: 2/5. Current State: 0
Time 405000: Cell (RESETTING) Delay: 3/5. Current State: 0
Time 415000: Cell (RESETTING->IDLE) COMPLETE. New State: 1, Endurance: 6
Time 425000: Write Cycle 4. Current State: 1, Busy: 0, Failed: 0
Time 425000: Cell (IDLE->SETTING) for target 0. Current: 1
Time 435000: Cell (SETTING) Delay: 0/5. Current State: 1
Time 445000: Cell (SETTING) Delay: 1/5. Current State: 1
Time 455000: Cell (SETTING) Delay: 2/5. Current State: 1
Time 465000: Cell (SETTING) Delay: 3/5. Current State: 1
Time 475000: Cell (SETTING->IDLE) COMPLETE. New State: 0, Endurance: 7
Time 485000: Write Cycle 5. Current State: 0, Busy: 0, Failed: 0
Time 485000: Cell FAILED! Endurance Limit (7) reached. Current endurance: 7
Time 485000: Cell (IDLE->RESETTING) for target 1. Current: 0
Time 495000: Cell (RESETTING) Delay: 0/5. Current State: 0
Time 505000: Cell (RESETTING) Delay: 1/5. Current State: 0
Time 515000: Cell (RESETTING) Delay: 2/5. Current State: 0
Time 525000: Cell (RESETTING) Delay: 3/5. Current State: 0
Time 535000: Cell (RESETTING->IDLE) COMPLETE. New State: 1, Endurance: 8
Time 545000: Write Cycle 6. Current State: 1, Busy: 0, Failed: 1
Time 545000: Cell FAILED at cycle 6. Remaining cycles will attempt writes to a failed cell.
Time 545000: Attempting write to a FAILED cell.
Time 545000: Cell (IDLE) Write attempt to FAILED cell. Current: 1, Endurance: 8
Time 555000: Cell (IDLE) Write attempt to FAILED cell. Current: 1, Endurance: 8
Time 565000: Cell (IDLE) Write attempt to FAILED cell. Current: 1, Endurance: 8
Time 575000: Cell (IDLE) Write attempt to FAILED cell. Current: 1, Endurance: 8
Time 585000: Cell (IDLE) Write attempt to FAILED cell. Current: 1, Endurance: 8
Time 595000: Cell (IDLE) Write attempt to FAILED cell. Current: 1, Endurance: 8
Time 605000: Cell (IDLE) Write attempt to FAILED cell. Current: 1, Endurance: 8
Time 615000: Cell (IDLE) Write attempt to FAILED cell. Current: 1, Endurance: 8
Time 625000: Cell (IDLE) Write attempt to FAILED cell. Current: 1, Endurance: 8
Time 635000: Cell (IDLE) Write attempt to FAILED cell. Current: 1, Endurance: 8
Time 645000: Write attempt on failed cell completed. Busy: 0, Failed: 1, Read Data: 1
----------------------------------------------------
           ReRAM Cell Testbench Finished
----------------------------------------------------
$finish called at time : 645 ns : File "C:/Users/hrish/Downloads/ReRAM_Thesis/Project
Folder/Sim/Sim.srcs/sim_1/new/reram_cell_tb.sv" Line 177
INFO: [USF-XSim-96] XSim completed. Design snapshot 'reram_cell_tb_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ms
launch_simulation: Time (s): cpu = 00:00:04 ; elapsed = 00:00:07 . Memory (MB): peak = 1342.566 ; gain
= 0.000
```


reram_core.sv
reram_cell_tb.sv
Untitled 2
0.000 ns
Name
Value
clk
0
rst_n
0
write_en
0
target_state
1
read_data
1
busy
0
failed
0
i[31:0]
XXXXXXXX
LRS_STATE_TB
0
HRS_STATE_TB
1
SET_DELAY...TB[31:0]
00000005
RESET_DE...B[31:0]
00000005
ENDURANC...TB[31:0]
00000007
CLOCK_PE...OD[31:0]
0000000a
0 ns
100 ns
200 ns
300 ns
400 ns
500 ns
600 ns
00000000
00000001
00000002
00000003
00000004
00000005
00000006


clk, rst_n, current=HRS,delay=0,endurance=0
init
fail_flag=true
error_state=true
def
write_en, fail_flag=false
IDLE
target=HRS, delay++,endurance++
WRITE done
WRITE done
target=LRS,delay++,endurance++
SETTING
RESETTING