Lab 1: Report on gen_tick

Table of Contents

Introduction

This report discusses the implementation and testing of the gen_tick module in Verilog. The gen_tick module generates a tick signal at a specified frequency based on a source clock input. The accompanying test bench verifies the functionality of the gen_tick module by simulating different scenarios and configurations.

gen_tick Module

The gen_tick module produces a tick signal at a given frequency (TICK_FREQ) from an input source clock (SRC_FREQ). The module includes parameters for the source clock frequency and the desired output tick frequency.

Module Code

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 06/06/2024 06:17:18 PM
// Design Name:
// Module Name: gen_tick
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
```

```
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

module gen_tick #(
    parameter SRC_FREQ = 5000,   // Source clock frequency in Hz
    parameter TICK_FREQ = 1      // Output tick frequency in Hz
)(
    input src_clk,    // Source clock input
    input enable,     // Enable signal
    output tick       // Output tick signal
);

// Calculate the number of source clock cycles per tick
localparam integer TICK_COUNT = SRC_FREQ / TICK_FREQ;

// Declare registers and wires here
reg [31:0] counter = 0;
reg tick_reg = 0;

always @(posedge src_clk) begin
    if (!enable) begin
        counter <= 0;
        tick_reg <= 0;  // No tick if not enabled
    end else begin
        if (counter == TICK_COUNT - 1) begin
            counter <= 0;
            tick_reg <= ~tick_reg;  // Toggle the tick signal
        end else begin
            counter <= counter + 1;
        end
    end
end

// Assign the tick register to the output tick signal
assign tick = tick_reg;

endmodule
```

Explanation

- **Parameters**:
    - SRC_FREQ: Frequency of the input source clock in Hz.
    - TICK_FREQ: Desired frequency of the output tick signal in Hz.
- **Local Parameters**:

    - TICK_COUNT: Number of source clock cycles required for one tick period (SRC_FREQ / TICK_FREQ).
- **Registers**:
    - counter: A 32-bit register to count the source clock cycles.
    - tick_reg: A register to hold the tick signal state.
- **Always Block**:
    - If enable is low, counter and tick_reg are reset to 0.
    - If enable is high, counter increments on each rising edge of src_clk.
    - When counter reaches TICK_COUNT - 1, it resets to 0 and toggles the tick_reg signal.
- **Output Assignment**:
    - The tick output is assigned the value of tick_reg.

Test Bench for gen_tick Module

The test bench lab01_tb is designed to verify the functionality of the gen_tick module. It instantiates three instances of gen_tick with different configurations and monitors their outputs.

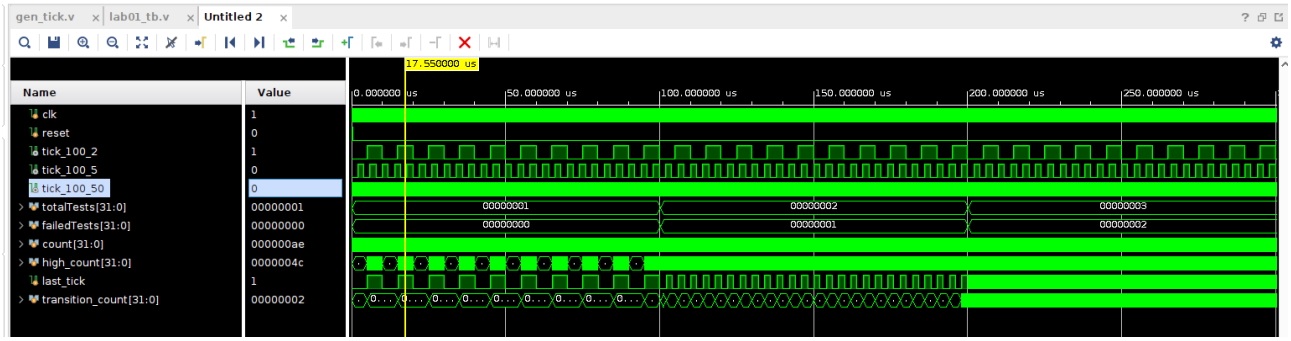
Code:

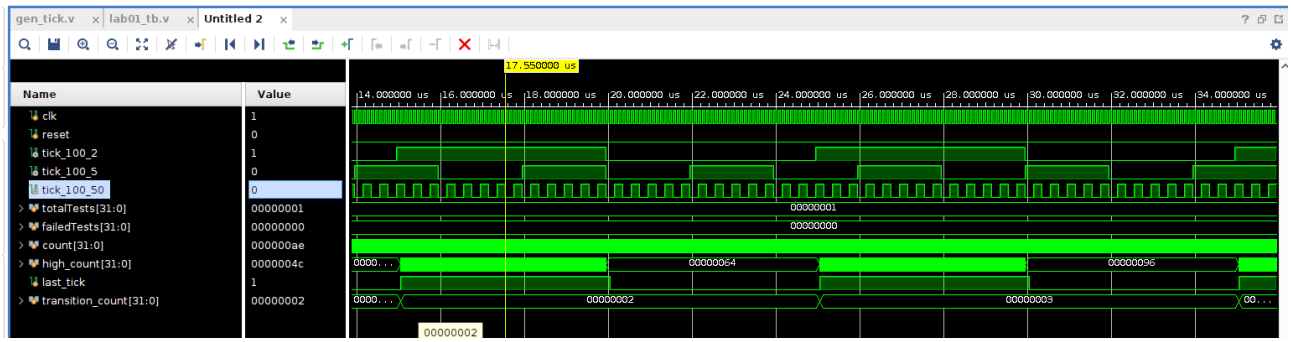
```
`timescale 1ns / 1ps

module lab01_tb;
   // Inputs
   reg clk;
   reg reset;

   // Outputs


   // ------------------------------------------------------
   // Setup output file for possible debugging uses
   // ------------------------------------------------------
   initial
   begin
      $dumpfile("lab01.vcd");
      $dumpvars(0);
   end

   // Declare wires for each unit under test
   wire tick_100_2;
   wire tick_100_5;
   wire tick_100_50;

   // ------------------------------------------------------
   // Instantiate at least 3 Units Under Test (UUTs)
```

```
// ------------------------------------------------------
gen_tick #(.SRC_FREQ(100), .TICK_FREQ(2)) uut_100_2 (
    .src_clk(clk),
    .enable(1'b1),
    .tick(tick_100_2)
);

gen_tick #(.SRC_FREQ(100), .TICK_FREQ(5)) uut_100_5 (
    .src_clk(clk),
    .enable(1'b1),
    .tick(tick_100_5)
);

gen_tick #(.SRC_FREQ(100), .TICK_FREQ(50)) uut_100_50 (
    .src_clk(clk),
    .enable(1'b1),
    .tick(tick_100_50)
);

initial begin
    clk = 0; reset = 1; #50;
    clk = 1; reset = 1; #50;
    clk = 0; reset = 0; #50;
    clk = 1; reset = 0; #50;

    forever begin
        clk = ~clk; #50;
    end
end

integer totalTests = 0;
integer failedTests = 0;

integer count = 0;
integer high_count = 0;
reg last_tick = 0;
integer transition_count = 0;

initial begin // Test suite
    // Reset
    @(negedge reset); // Wait for reset to be released (from another initial block)
    @(posedge clk);   // Wait for first clock out of reset
    #10; // Wait 10 cycles

    // ---------------------------------------------
    // Testing Source clock 100Hz, Tick 2Hz
    // ---------------------------------------------
    $write("Test Source clock 100Hz, Tick 2Hz ... ");
    totalTests = totalTests + 1;
    while(count < 1000) begin
        @(posedge clk);
        if (last_tick == 0 & tick_100_2 != last_tick) begin
```

```
        transition_count = transition_count + 1;
    end
    count = count + 1;
    if (tick_100_2 == 1) begin
        high_count = high_count + 1;
    end
    last_tick = tick_100_2;
end

if (high_count == 500 & transition_count == 20) begin
    $display("PASSED");
end else begin
    $display("FAILED");
    failedTests = failedTests + 1;
end
$display("Load (%d/%d): %0.2f", high_count, count, 1.0 * high_count / count);
$display("Transition count: %d", transition_count);

// Re-initialize counters for each test
last_tick = 0;
transition_count = 0;
count = 0;
high_count = 0;

// ----------------------------------------------
// Testing Source clock 100Hz, Tick 5Hz
// ----------------------------------------------
$write("Test Source clock 100Hz, Tick 5Hz ... ");
totalTests = totalTests + 1;
while(count < 1000) begin
    @(posedge clk);
    if (last_tick == 0 & tick_100_5 != last_tick) begin
        transition_count = transition_count + 1;
    end
    count = count + 1;
    if (tick_100_5 == 1) begin
        high_count = high_count + 1;
    end
    last_tick = tick_100_5;
end

if (high_count == 1000 & transition_count == 50) begin
    $display("PASSED");
end else begin
    $display("FAILED");
    failedTests = failedTests + 1;
end
$display("Load (%d/%d): %0.2f", high_count, count, 1.0 * high_count / count);
$display("Transition count: %d", transition_count);

// Re-initialize counters for each test
last_tick = 0;
```

```
        transition_count = 0;
        count = 0;
        high_count = 0;

        // ------------------------------------------
        // Testing Source clock 100Hz, Tick 50Hz
        // ------------------------------------------
        $write("Test Source clock 100Hz, Tick 50Hz ... ");
        totalTests = totalTests + 1;
        while(count < 1000) begin
            @(posedge clk);
            if (last_tick == 0 & tick_100_50 != last_tick) begin
                transition_count = transition_count + 1;
            end
            count = count + 1;
            if (tick_100_50 == 1) begin
                high_count = high_count + 1;
            end
            last_tick = tick_100_50;
        end

        if (high_count == 500 & transition_count == 100) begin
            $display("PASSED");
        end else begin
            $display("FAILED");
            failedTests = failedTests + 1;
        end
        $display("Load (%d/%d): %0.2f", high_count, count, 1.0 * high_count / count);
        $display("Transition count: %d", transition_count);

        $finish;
    end
endmodule
```





Explanation

- **Inputs**:
  - clk: Clock signal for the test bench.
  - reset: Reset signal for the test bench.
- **Initialization**:
  - $dumpfile and $dumpvars are used for waveform generation.
  - Three instances of the gen_tick module are instantiated with different configurations:
    - uut_100_2 with SRC_FREQ = 100 Hz and TICK_FREQ = 2 Hz.
    - uut_100_5 with SRC_FREQ = 100 Hz and TICK_FREQ = 5 Hz.
    - uut_100_50 with SRC_FREQ = 100 Hz and TICK_FREQ = 50 Hz.
- **Clock Generation**:
  - The clock signal (clk) is toggled every 50 time units to simulate a 10 MHz clock.
- **Test Scenarios**:
  - Each test case verifies the tick generation for different configurations by counting the high ticks and transitions over 1000 clock cycles.
  - **Test Case 1**: SRC_FREQ = 100 Hz, TICK_FREQ = 2 Hz.
    - The expected number of high ticks is 500, and the number of transitions should be 20.
  - **Test Case 2**: SRC_FREQ = 100 Hz, TICK_FREQ = 5 Hz.
    - The expected number of high ticks is 1000, and the number of transitions should be 50.
  - **Test Case 3**: SRC_FREQ = 100 Hz, TICK_FREQ = 50 Hz.
    - The expected number of high ticks is 500, and the number of transitions should be 100.
- **Output Monitoring**:
  - The $display task is used to print the test results, indicating whether each test passed or failed based on the expected output.

Conclusion

The gen_tick module successfully generates a tick signal at a specified frequency derived from a source clock. The test bench verifies the functionality of the gen_tick module by simulating various

configurations and observing the output behavior. The simulation results confirm that the module operates correctly when tested with different source clock and tick frequencies.