Question 6

Objective:

The goal was to design a timer using a 50 MHz oscillator that counts down from a specified time (minutes and seconds) and asserts a done signal when the timer expires. The design incorporates an active-low synchronous reset and was validated using a testbench.

Design:

Clock and Reset: The timer operates on a 50 MHz clock (clk) and uses an active-low synchronous reset (reset_n).

Inputs: Two 6-bit inputs (minutes and seconds) set the initial countdown time.

Outputs:

done: Asserted when the timer reaches zero.

count_minutes and count_seconds: Display the remaining time during countdown.

Microsecond Pulse Generation: A 1 µs pulse is generated from the 50 MHz clock to drive the countdown logic. This ensures precise timing and better simulation granularity.

The code:

```
`timescale 1ns / 1ps

module timer(

    input clk,                    // 50 MHz clock input

    input reset_n,                // Active-low synchronous reset

    input [5:0] minutes,          // 6-bit input for minutes (0 to 59)

    input [5:0] seconds,          // 6-bit input for seconds (0 to 59)

    output reg done,

    output reg [5:0] count_minutes,

    output reg [5:0] count_seconds

);


    reg [5:0] microsec_counter;   // Counter for generating 1 microsecond pulse
```

```
    reg microsec_pulse;           // 1 microsecond pulse for each tick


    // Generate 1 microsecond pulse from 50 MHz clock
    always @(posedge clk or negedge reset_n) begin
        if (!reset_n) begin
            microsec_counter <= 0;
            microsec_pulse <= 0;
        end else begin
            if (microsec_counter == 25 - 1) begin  // 25 clock cycles = 1 microsecond
                microsec_counter <= 0;
                microsec_pulse <= 1;  // Generate 1 microsecond pulse
            end else begin
                microsec_counter <= microsec_counter + 1;
                microsec_pulse <= 0;
            end
        end
    end


    // Active-low synchronous reset and countdown logic using microsecond pulse
    always @(posedge microsec_pulse or negedge reset_n) begin
        if (!reset_n) begin
            count_minutes <= minutes;
            count_seconds <= seconds;
            done <= 0;
        end else begin
            // Countdown logic
            if (count_minutes == 0 && count_seconds == 0) begin
```

```
                done <= 1;  // Timer has expired
            end else begin
                done <= 0;
                if (count_seconds == 0) begin
                    if (count_minutes > 0) begin
                        count_minutes <= count_minutes - 1;
                        count_seconds <= 59;
                    end
                end else begin
                    count_seconds <= count_seconds - 1;
                end
            end
        end
    end
endmodule
```

Simulation:

Testbench: The testbench initializes the timer to 1 minute and 5 seconds, applies a reset, and starts the countdown. After 1 minute and 5 seconds, it checks if the done signal is asserted.

Result: The simulation confirmed that the timer works as expected, with the done signal correctly asserted at the end of the countdown.

Reason for Using Microsecond Granularity:

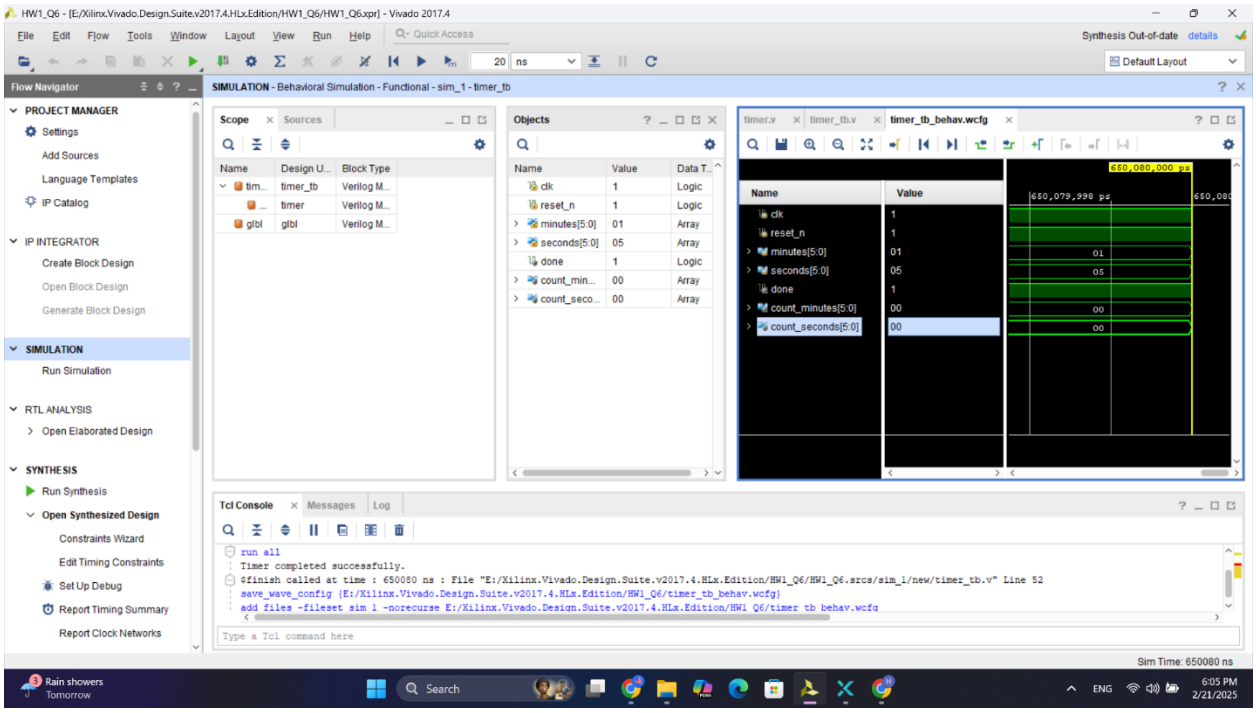
Simulation Accuracy: Using a 1 µs pulse instead of a 1-second pulse allows for finer granularity in simulation, making it easier to debug and verify the design. It also ensures that the countdown logic is tested more rigorously, as it operates at a higher frequency.

Efficiency: Simulating at a microsecond level is computationally efficient compared to simulating at a second level, especially for shorter durations.

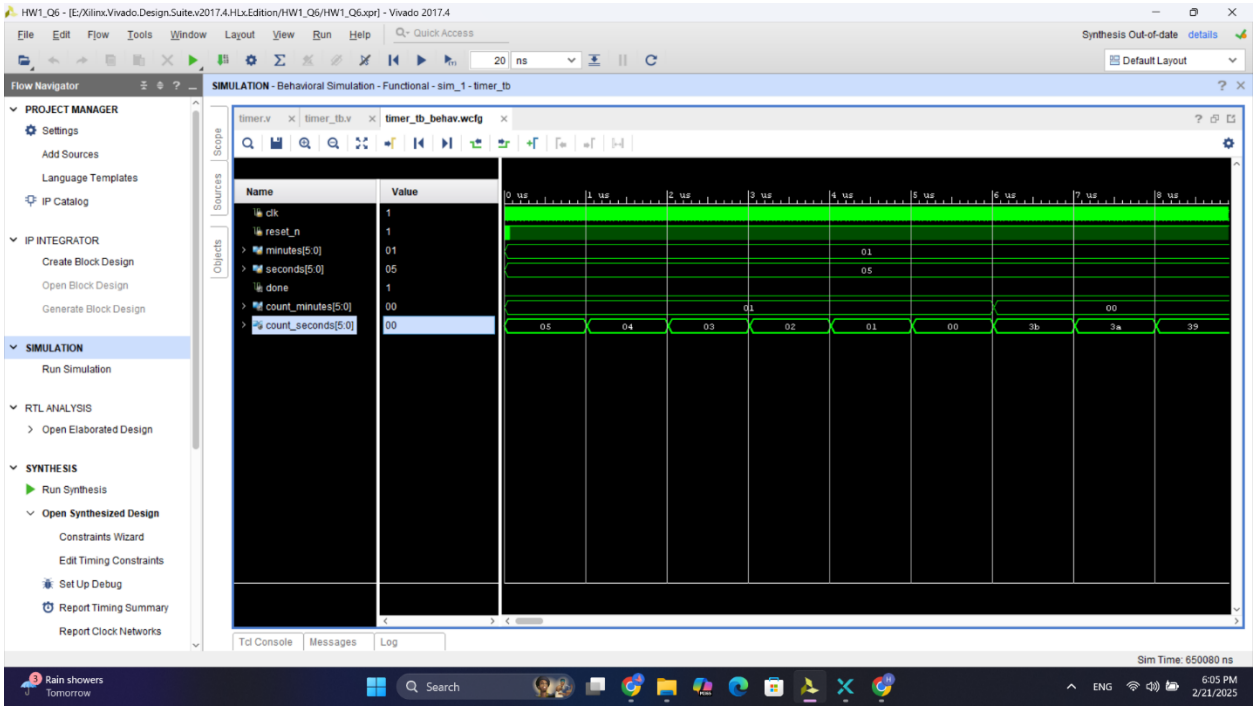
Test bench code:

```
`timescale 1ns / 1ps

module timer_tb;

    reg clk;

reg reset_n;

reg [5:0] minutes;

reg [5:0] seconds;

wire done;

wire [5:0] count_minutes;

wire [5:0] count_seconds;


// Instantiate the timer module

timer uut (

    .clk(clk),

    .reset_n(reset_n),

    .minutes(minutes),

    .seconds(seconds),

    .done(done),

    .count_minutes(count_minutes),

    .count_seconds(count_seconds)

);


// Generate a 50 MHz clock

always begin

    #20 clk = ~clk;  // Toggle clk every 20ns (50 MHz)

end
```

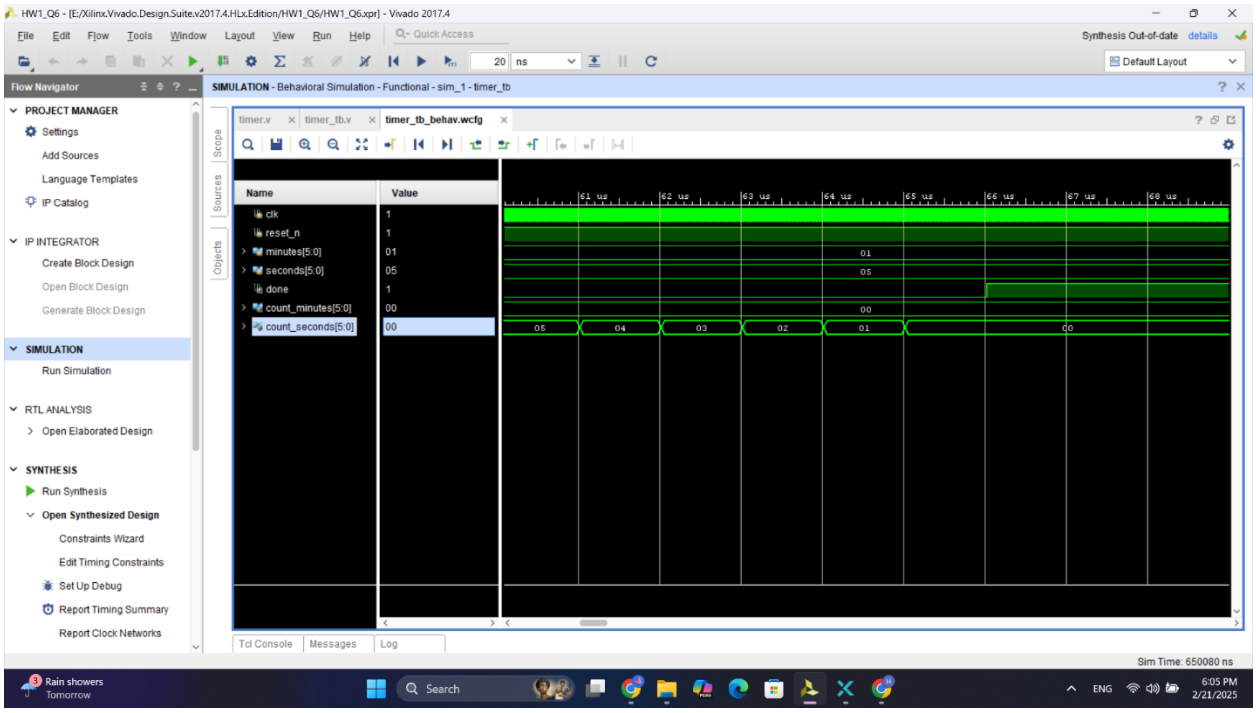
```
// Test sequence
initial begin
    // Initialize signals
    clk = 0;
    reset_n = 0;
    minutes = 6'd1;  // Set initial time to 1 minute 5 seconds
    seconds = 6'd5;

    // Apply reset
    #20 reset_n = 1;

    // Apply reset and then start the countdown
    #20 reset_n = 0; // Assert reset
    #20 reset_n = 1; // Deassert reset to start timer

    // Wait for the timer to expire (i.e., 1 minute 5 seconds)
    #650000;
    // Check if the done signal is raised
    if (done) begin
        $display("Timer completed successfully.");
    end else begin
        $display("Timer did not complete correctly.");
    end
    // Finish simulation
    #20 $finish;
end
endmodule
```



as we can see in the console the timer completed successfully



as we can see in the wave form the counter is decrementing every 1 microsecond.



As we can see in the wave form when the counter reaches 0 done is raised to 1.