EE5260: HARDWARE DESCRIPTION LANGUAGE

Take Home Assignment 02

NAME : MUNSIF M.F.A

REG No. : EG/ 2021/ 4684

SEMESTER: 05

DATE : 22 /10 /2024

Task 01:



module IC7490 (

    input wire clk,    // Clock input

    input wire reset,  // Reset input

    output reg [3:0] Q // BCD output (QD, QC, QB, QA)

);

    // Always block for counting functionality

    always @(posedge clk or posedge reset) begin

        if (reset) begin

            Q <= 4'b0000; // Reset to 0

        end else begin

            if (Q == 4'b1001) begin

                Q <= 4'b0000; // Roll back to 0 after reaching 9

            end else begin

                Q <= Q + 1;   // Increment the count

            end

        end

    end

endmodule



module IC7490TB;

    reg clk;

    reg reset;

    wire [3:0] Q;

    IC7490 uut (

        .clk(clk),

        .reset(reset),

        .Q(Q)

    );

    initial begin

        $dumpfile("counter.vcd");

        $dumpvars(0, IC7490TB);

        clk = 0;

        forever #5 clk = ~clk; // Toggle every 5ns

    end

    initial begin

        // Monitor output

        $monitor("Time = %0t | Reset = %b | Count = %b", $time, reset, Q);

        reset = 1;

        #10 reset = 0;

        #100;

        #10 reset = 1;

        #10 reset = 0;

        #100;

        $finish;

    end

endmodule

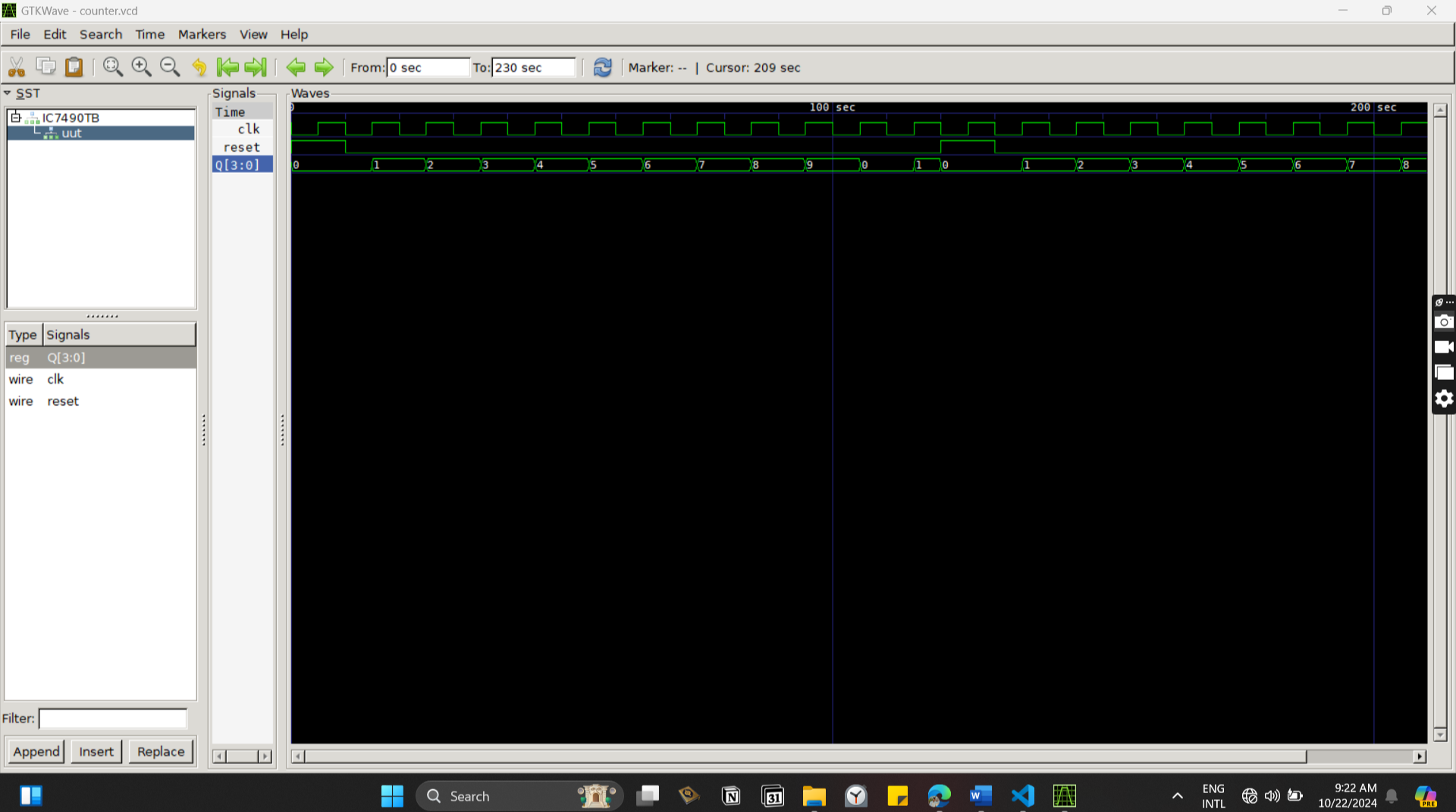


Figure : IC7490TB output which displays the BCD Count Sequence

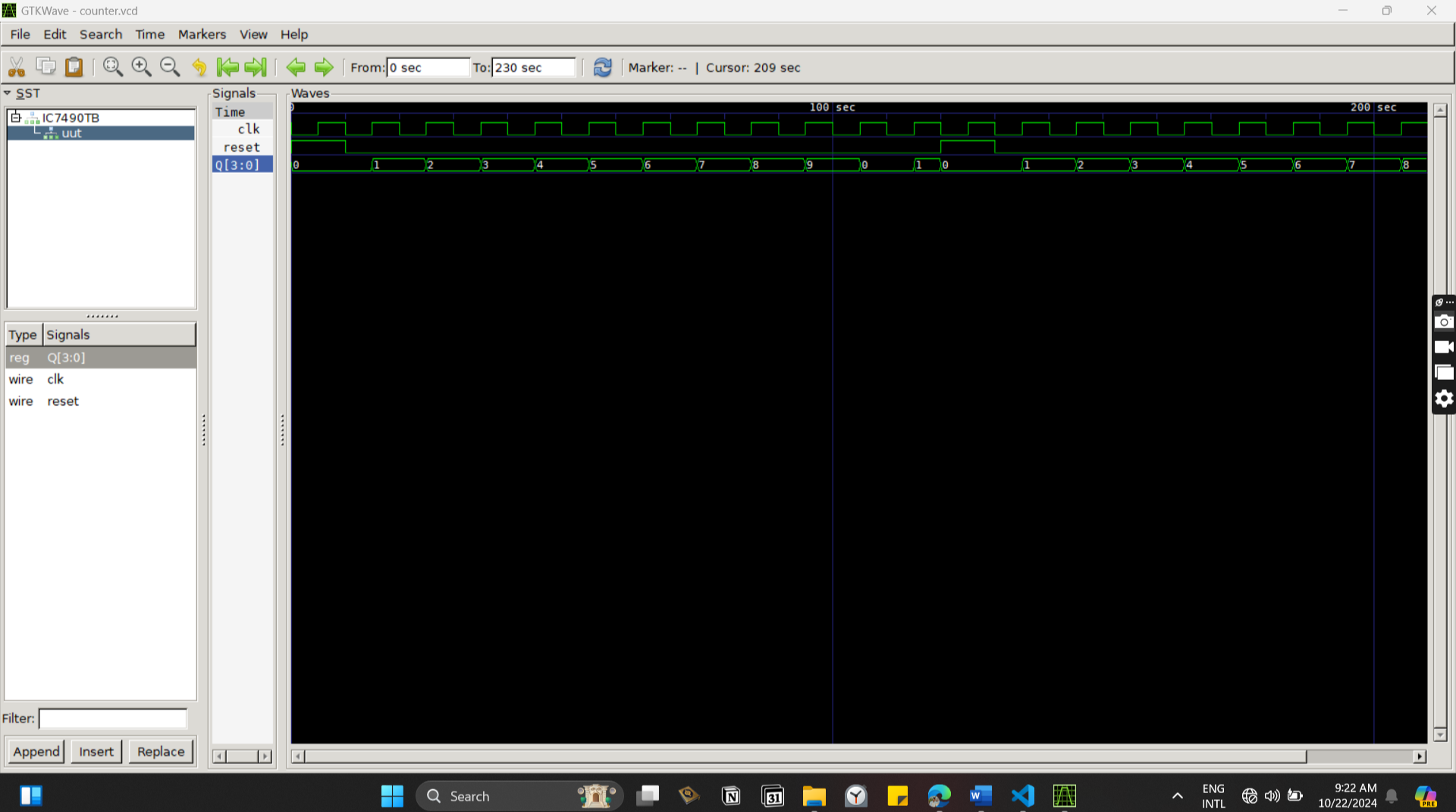


Figure : IC7490TB output - zoomed view

Task02:



module Decoder (

    input wire [3:0] BCD, // 4-bit BCD input

    output reg [6:0] seg  // 7-segment output (a to g)

);

    always @(BCD) begin

        case (BCD)

            4'b0000: seg = 7'b0111111;  // 0

            4'b0001: seg = 7'b0000110;  // 1

            4'b0010: seg = 7'b1011011;  // 2

            4'b0011: seg = 7'b1001111;  // 3

            4'b0100: seg = 7'b1100110;  // 4

            4'b0101: seg = 7'b1101101;  // 5

            4'b0110: seg = 7'b1111101;  // 6

            4'b0111: seg = 7'b0000111;  // 7

            4'b1000: seg = 7'b1111111;  // 8

            4'b1001: seg = 7'b1101111;  // 9

            default: seg = 7'b0000000;  // For Error Values

        endcase

    end

endmodule



module IC4017 (

    input wire clk,     // Clock input

    input wire reset,   // Reset input

    output reg [9:0] Q  // 10 output lines (one per count)

);

    reg [3:0] count;    // 4-bit counter to track the count (0-9)

    always @(posedge clk or posedge reset) begin

        if (reset) begin

            count <= 4'b0000; // Reset count to 0

            Q <= 10'b0000000001; // Set the first output HIGH

        end else begin

            if (count == 4'b1001) begin

                count <= 4'b0000; // Reset count after 9

            end else begin

                count <= count + 1; // Increment the count

            end

            // Shift the output Q to reflect the current count

            Q <= (10'b1 << count); // Set only one bit HIGH at a time

        end

    end

endmodule

Task03:



module Counter (

    input wire clk,        // Clock input

    input wire reset,      // Reset input

    output wire [6:0] seg\_tens,  // 7-segment display for tens

    output wire [6:0] seg\_ones   // 7-segment display for ones

);

    reg [5:0] count; // 6-bit counter to count up to 40

    wire [3:0] tens, ones; // BCD digits for tens and ones

    // Increment the counter on each clock pulse

    always @(posedge clk or posedge reset) begin

        if (reset) begin

            count <= 6'b0; // Reset the counter to 0

        end else if (count == 6'd40) begin

            count <= 6'b0; // Reset the counter when it reaches P (40)

        end else begin

            count <= count + 1; // Increment the counter

        end

    end

    // Convert the count into BCD digits for tens and ones

    assign tens = count / 10; // Tens digit

    assign ones = count % 10; // Ones digit

    // Use the Decoder module to drive the 7-segment displays

    Decoder decode\_tens(.BCD(tens), .seg(seg\_tens));

    Decoder decode\_ones(.BCD(ones), .seg(seg\_ones));

endmodule



module CounterTB();

    reg clk;       // Testbench clock

    reg reset;     // Testbench reset

    wire [6:0] seg\_tens; // Tens 7-segment output

    wire [6:0] seg\_ones; // Ones 7-segment output

    // Instantiate the Counter module

    Counter counter\_inst (

        .clk(clk),

        .reset(reset),

        .seg\_tens(seg\_tens),

        .seg\_ones(seg\_ones)

    );

    // Clock generation

    initial begin

        clk = 0;

        forever #5 clk = ~clk; // Toggle the clock every 5 time units

        $dumpfile("counter\_wave.vcd");

        $dumpvars(0, CounterTB);

    end

    // Simulation procedure

    initial begin

        reset = 1;     // Start with reset active

        #10 reset = 0; // Deactivate reset after 10 time units

        // Let the counter run for a few clock cycles until it counts to 40

        #415;

        $finish;       // End the simulation

    end

    // Monitoring the output

    initial begin

        $monitor("Time = %0t, Tens = %b, Ones = %b", $time, seg\_tens, seg\_ones);

    end

endmodule

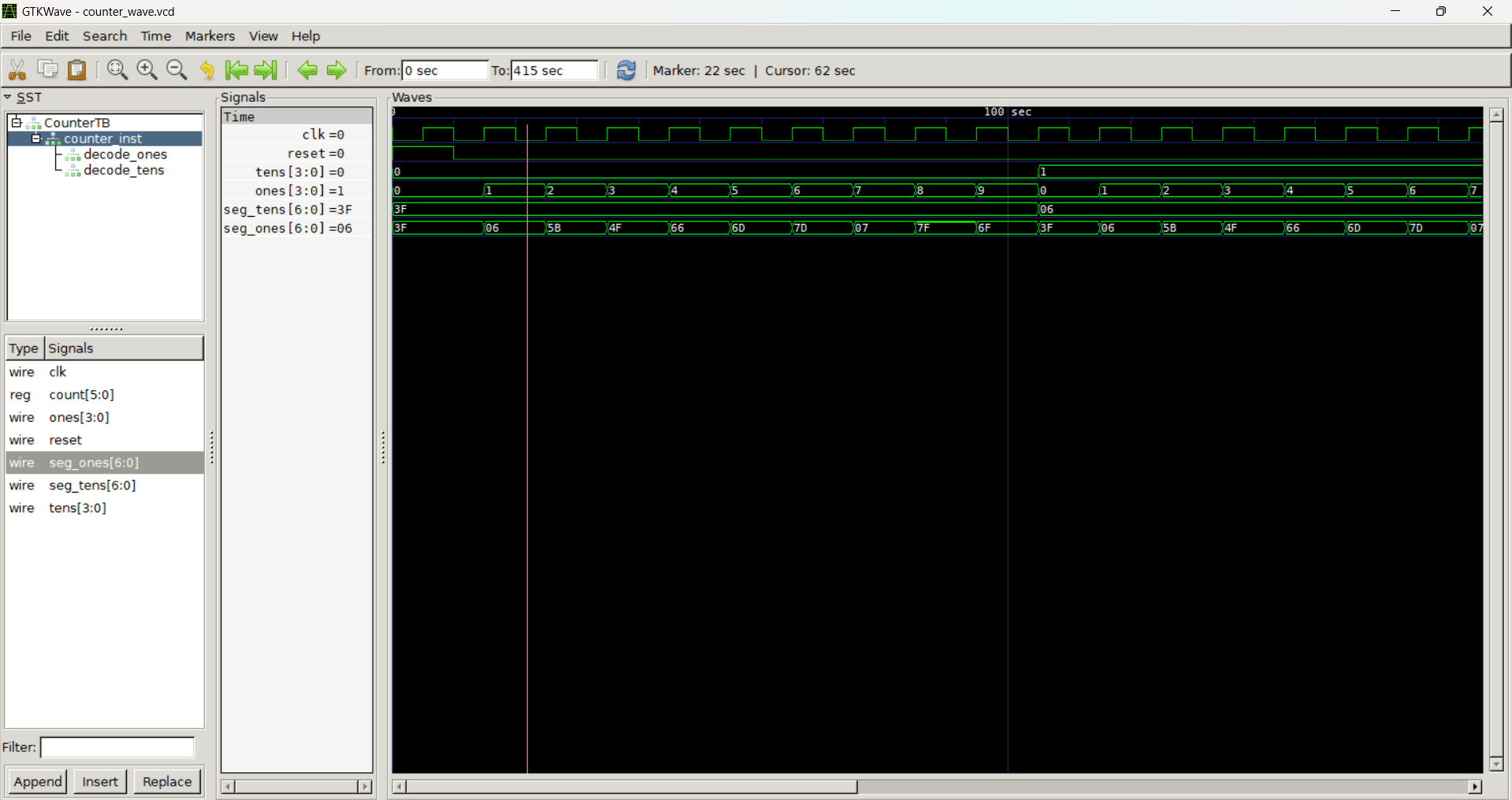


Figure : CounterTB module’s output count from 0 to 17

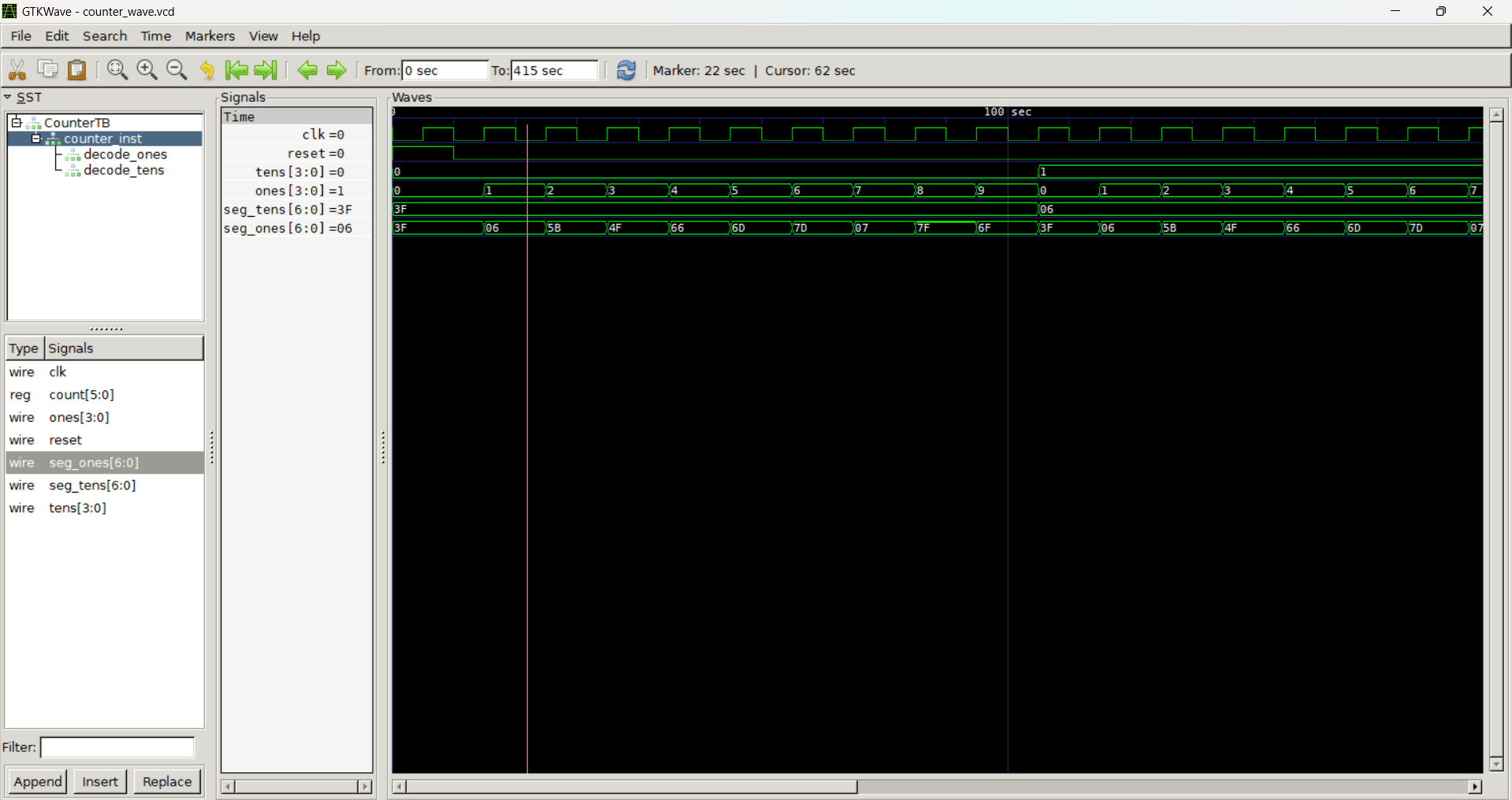


Figure : CounterTB module’s output - zoomed view

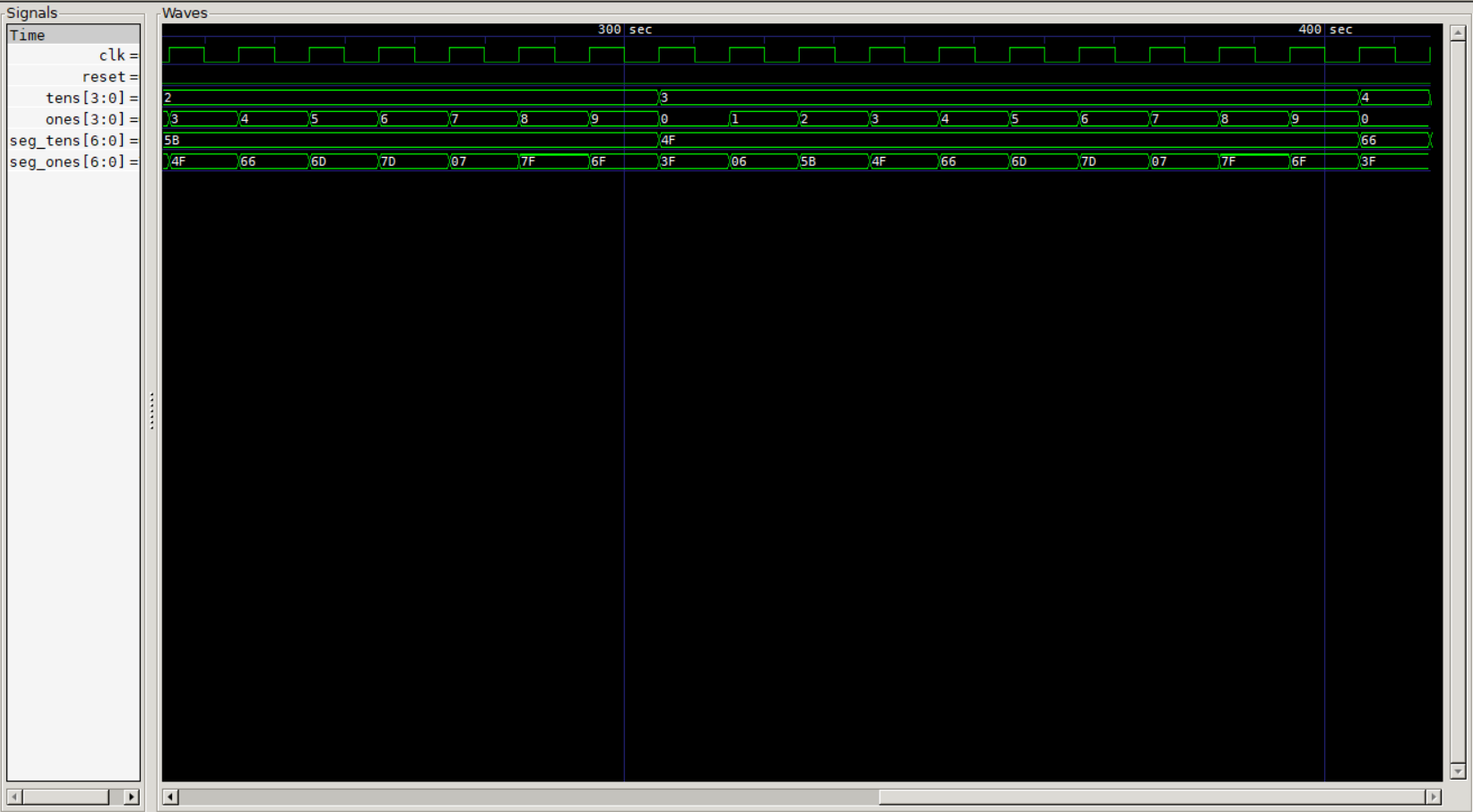


Figure : CounterTB module’s output until 40 as my final digit of the index number is 4

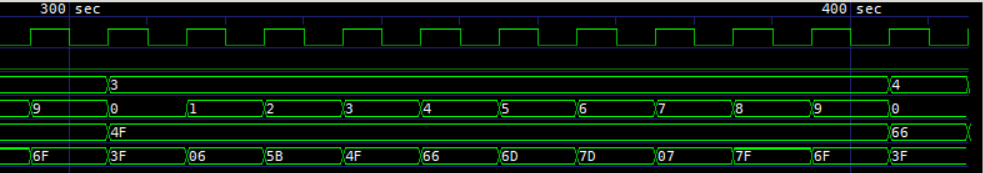


Figure : CounterTB module’s output up to 40 - zoomed view