# Real Time Clock

## RTL CODE:

```
module seconds_counter (
    input clk,          // 1 Hz clock input
    input reset,        // Synchronous reset
    output reg [3:0] SEC_L, // Seconds LSB (units place: 0?9)
    output reg [3:0] SEC_M, // Seconds MSB (tens place: 0?5)
    output reg sec_tick     // Goes HIGH when seconds = 59 and resets to 00
);
always @(posedge clk) begin
    if (reset) begin
        SEC_L <= 4'd0;
        SEC_M <= 4'd0;
        sec_tick <= 1'b0;
    end else begin
        sec_tick <= 1'b0;  // Default to 0 each clock

        if (SEC_L == 4'd9) begin
            SEC_L <= 4'd0;

            if (SEC_M == 4'd5) begin
                SEC_M <= 4'd0;
                sec_tick <= 1'b1; // One full minute passed
            end else begin
                SEC_M <= SEC_M + 1'b1;
            end

        end else begin
            SEC_L <= SEC_L + 1'b1;
        end
    end
```

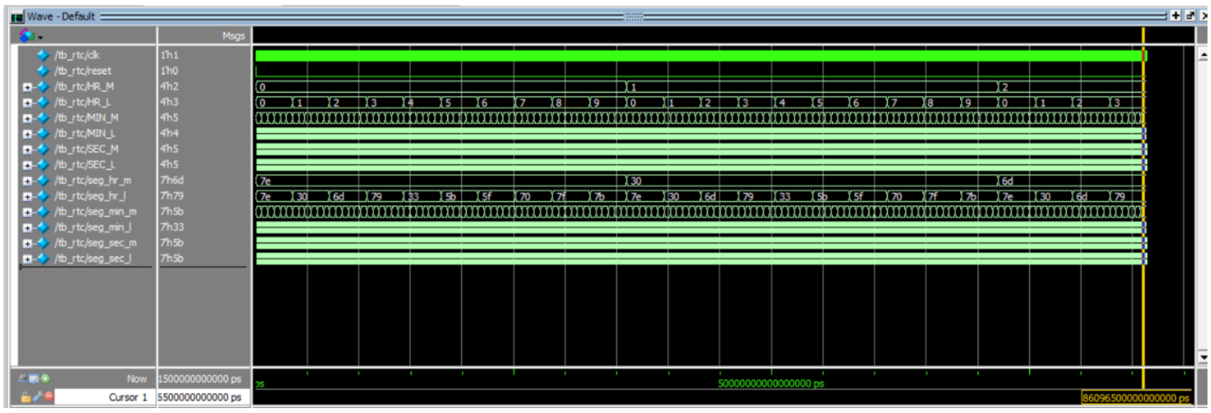
```
end
endmodule


module minutes_counter (
    input        clk,        // 1 Hz clock
    input        reset,      // synchronous reset
    input        sec_tick,   // pulse from seconds_counter when seconds roll over
    output reg [3:0] MIN_L,  // Minutes LSB (0?9)
    output reg [3:0] MIN_M,  // Minutes MSB (0?5)
    output reg      min_tick // goes HIGH for one clk when minutes roll over
);
always @(posedge clk) begin
    if (reset) begin
        MIN_L    <= 4'd0;
        MIN_M    <= 4'd0;
        min_tick <= 1'b0;
    end else begin
        // default no rollover
        min_tick <= 1'b0;


        if (sec_tick) begin
            // only update minutes when sec_tick pulses
            if (MIN_L == 4'd9) begin
                MIN_L <= 4'd0;
                if (MIN_M == 4'd5) begin
                    MIN_M    <= 4'd0;
                    min_tick <= 1'b1;  // one full hour (minute rollover)
                end else begin
                    MIN_M <= MIN_M + 1'b1;
                end
            end else begin
                MIN_L <= MIN_L + 1'b1;
            end
        end
    end
end
endmodule


module hours_counter (
    input        clk,       // 1 Hz clock
    input        reset,     // synchronous reset
    input        min_tick,  // pulse when minutes roll over 59?00
    output reg [3:0] HR_L,  // Hours LSB (units: 0?9 or 0?3 when in 20?s)
    output reg [3:0] HR_M,  // Hours MSB (tens: 0?2)
    output reg      hr_tick // pulses HIGH when hours roll over 23?00
);
always @(posedge clk) begin
    if (reset) begin
        HR_L    <= 4'd0;
        HR_M    <= 4'd0;
        hr_tick <= 1'b0;
    end else begin
        // default: no hour rollover
        hr_tick <= 1'b0;

        if (min_tick) begin
            // Case A: in ?20?s? (HR_M == 2), units go 0?3
            if (HR_M == 4'd2) begin
                if (HR_L == 4'd3) begin
                    HR_L    <= 4'd0;
                    HR_M    <= 4'd0;
                    hr_tick <= 1'b1;  // rolled over past 23
                end else begin
                    HR_L <= HR_L + 1'b1;
                end
            end
            // Case B: in ?0?s? or ?10?s? (HR_M == 0 or 1), units go 0?9
            else begin
```

```
              if (HR_L == 4'd9) begin
                 HR_L <= 4'd0;
                 HR_M <= HR_M + 1'b1;
              end else begin
                 HR_L <= HR_L + 1'b1;
              end
           end
        end
     end
  end
endmodule

module rtc_top (
    input        clk,       // 1 Hz system clock
    input        reset,     // synchronous reset
    // BCD outputs for display (HH:MM:SS)
    output [3:0] HR_M,      // hours tens (0?2)
    output [3:0] HR_L,      // hours units (0?9 or 0?3)
    output [3:0] MIN_M,     // minutes tens (0?5)
    output [3:0] MIN_L,     // minutes units (0?9)
    output [3:0] SEC_M,     // seconds tens (0?5)
    output [3:0] SEC_L      // seconds units (0?9)
    // optional tick outputs if you need them:
    // output sec_tick,
    // output min_tick,
    // output hr_tick
);
// Internal wires for ticks
wire sec_tick;
wire min_tick;
wire hr_tick;
// Instantiate seconds counter
seconds_counter u_sec (
    .clk      (clk),
    .reset    (reset),
    .SEC_L    (SEC_L),
    .SEC_M    (SEC_M),
    .sec_tick (sec_tick)
);
// Instantiate minutes counter
minutes_counter u_min (
    .clk      (clk),
    .reset    (reset),
    .sec_tick (sec_tick),
    .MIN_L    (MIN_L),
    .MIN_M    (MIN_M),
    .min_tick (min_tick)
);
// Instantiate hours counter
hours_counter u_hour (
    .clk      (clk),
    .reset    (reset),
    .min_tick (min_tick),
    .HR_L     (HR_L),
    .HR_M     (HR_M),
    .hr_tick  (hr_tick)
);
endmodule
```

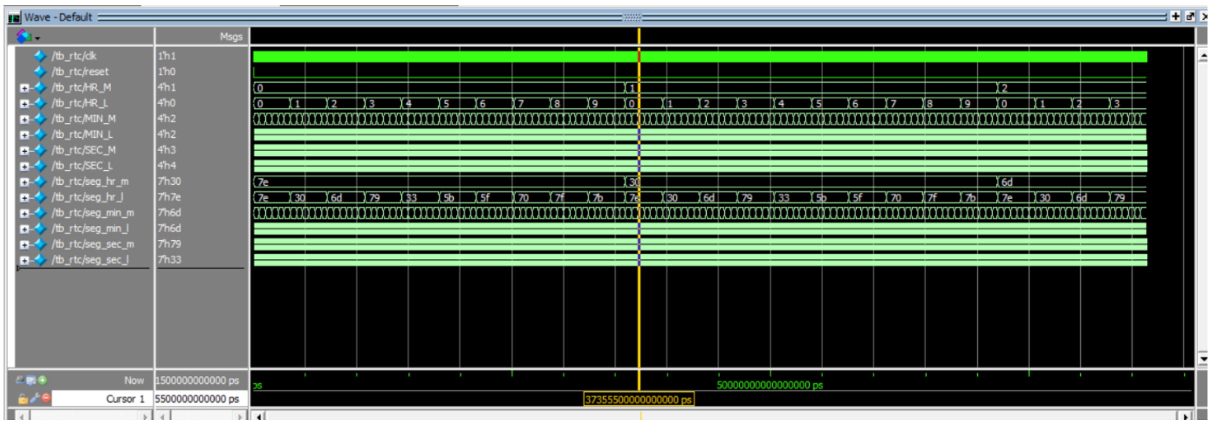
## TEST BENCH:

```
`timescale 1ns / 1ps
module tb_rtc;
  // Clock and reset
  reg clk;
  reg reset;
  // BCD outputs from rtc_top
  wire [3:0] HR_M, HR_L;
  wire [3:0] MIN_M, MIN_L;
  wire [3:0] SEC_M, SEC_L;
  // 7?segment outputs
  wire [6:0] seg_hr_m, seg_hr_l;
  wire [6:0] seg_min_m, seg_min_l;
  wire [6:0] seg_sec_m, seg_sec_l;
  // Instantiate the RTC
  rtc_top dut (
    .clk   (clk),
    .reset (reset),
    .HR_M  (HR_M),
    .HR_L  (HR_L),
    .MIN_M (MIN_M),
    .MIN_L (MIN_L),
    .SEC_M (SEC_M),
    .SEC_L (SEC_L)
  );
  // Instantiate BCD to 7 segment decoders
  bcd_to_7seg dec_hr_m (.bcd(HR_M),  .seg(seg_hr_m));
  bcd_to_7seg dec_hr_l (.bcd(HR_L),  .seg(seg_hr_l));
  bcd_to_7seg dec_min_m(.bcd(MIN_M), .seg(seg_min_m));
  bcd_to_7seg dec_min_l(.bcd(MIN_L), .seg(seg_min_l));
  bcd_to_7seg dec_sec_m(.bcd(SEC_M), .seg(seg_sec_m));
  bcd_to_7seg dec_sec_l(.bcd(SEC_L), .seg(seg_sec_l));
  // 1 Hz clock: toggle every 0.5 s = 500 000 000 ns
  initial clk = 0;
  always #500_000_000 clk = ~clk;
  // Reset and run for N seconds by counting clock edges
  initial begin
    reset = 1;
    // Wait two rising edges to ensure reset spans >1 cycle
    @(posedge clk);
    @(posedge clk);
    reset = 0;
    // Run for 86400 seconds (count 86400 rising edges of clk)
    repeat (86400)@(posedge clk);
    $finish;
  end
  // Console Monitor
  initial begin
    $monitor("Time = %0t | %d%d:%d%d:%d%d",
            $time, HR_M, HR_L, MIN_M, MIN_L, SEC_M, SEC_L);
  end
endmodule

// BCD to 7 Segment Decoder Module
module bcd_to_7seg (
    input  [3:0] bcd,
    output reg [6:0] seg
);
always @(*) begin
    case (bcd)
        4'd0: seg = 7'b1111110;
        4'd1: seg = 7'b0110000;
        4'd2: seg = 7'b1101101;
        4'd3: seg = 7'b1111001;
        4'd4: seg = 7'b0110011;
        4'd5: seg = 7'b1011011;
        4'd6: seg = 7'b1011111;
        4'd7: seg = 7'b1110000;
        4'd8: seg = 7'b1111111;
        4'd9: seg = 7'b1111011;
        default: seg = 7'b0000000;
    endcase
end
endmodule
```

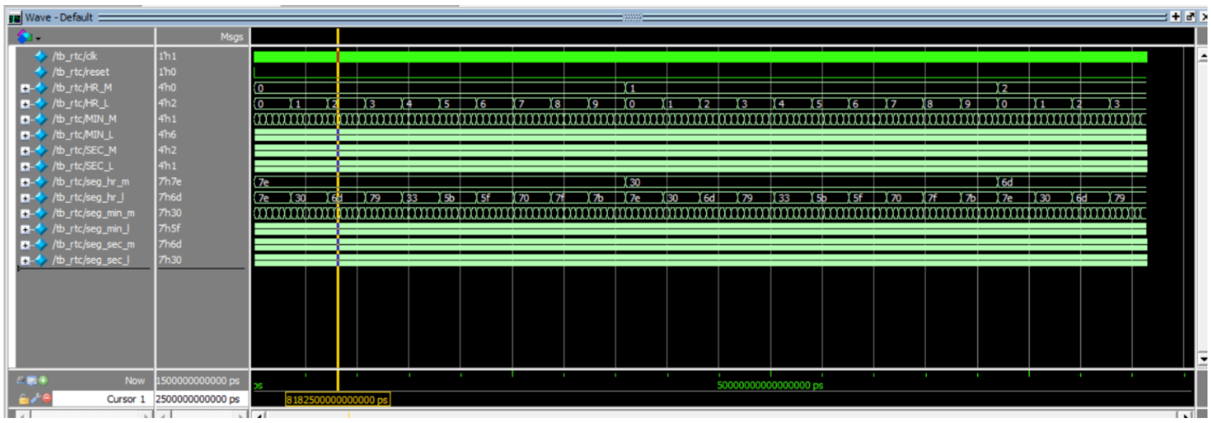
## OUTPUT:



```
Wave - Default
                        Msgs
/tb_rtc/clk             1'h1
/tb_rtc/reset           1'h0
/tb_rtc/HR_M            4'h2
/tb_rtc/HR_L            4'h3
/tb_rtc/MIN_M           4'h5
/tb_rtc/MIN_L           4'h4
/tb_rtc/SEC_M           4'h5
/tb_rtc/SEC_L           4'h5
/tb_rtc/seg_hr_m        7'h6d
/tb_rtc/seg_hr_l        7'h79
/tb_rtc/seg_min_m       7'h5b
/tb_rtc/seg_min_l       7'h33
/tb_rtc/seg_sec_m       7'h5b
/tb_rtc/seg_sec_l       7'h5b
Now       1500000000000 ps
Cursor 1  5500000000000 ps
```



```
Wave - Default
                        Msgs
/tb_rtc/clk             1'h1
/tb_rtc/reset           1'h0
/tb_rtc/HR_M            4'h1
/tb_rtc/HR_L            4'h0
/tb_rtc/MIN_M           4'h2
/tb_rtc/MIN_L           4'h2
/tb_rtc/SEC_M           4'h3
/tb_rtc/SEC_L           4'h4
/tb_rtc/seg_hr_m        7'h30
/tb_rtc/seg_hr_l        7'h7e
/tb_rtc/seg_min_m       7'h6d
/tb_rtc/seg_min_l       7'h6d
/tb_rtc/seg_sec_m       7'h79
/tb_rtc/seg_sec_l       7'h33
Now       1500000000000 ps
Cursor 1  5500000000000 ps
```



```
Wave - Default
                        Msgs
/tb_rtc/clk             1'h1
/tb_rtc/reset           1'h0
/tb_rtc/HR_M            4'h0
/tb_rtc/HR_L            4'h2
/tb_rtc/MIN_M           4'h1
/tb_rtc/MIN_L           4'h6
/tb_rtc/SEC_M           4'h2
/tb_rtc/SEC_L           4'h1
/tb_rtc/seg_hr_m        7'h7e
/tb_rtc/seg_hr_l        7'h6d
/tb_rtc/seg_min_m       7'h30
/tb_rtc/seg_min_l       7'h5f
/tb_rtc/seg_sec_m       7'h6d
/tb_rtc/seg_sec_l       7'h30
Now       1500000000000 ps
Cursor 1  2500000000000 ps
```