perplexity

now add the seconds counter and full 6-digit 7-segment display logic with multiplexing

Based on the provided image and search results, I'll extend the real-time clock code with seconds counter functionality and full 6-digit 7-segment display multiplexing. Here's the complete code:

```
module clock(
    input clk, rst,
    output [6:0] Segment,
    output [5:0] Comnd
);
    // Counter registers
    reg [25:0] count;      // Main counter for 1-second timebase
    reg [5:0] scount;      // Seconds counter (0-59)
    reg [5:0] mcount;      // Minutes counter (0-59)
    reg [4:0] hcount;      // Hours counter (0-23)

    // Clear signals for counters
    wire clr_cnt;          // Clear main counter
    wire clr_s;            // Clear seconds counter
    wire clr_m;            // Clear minutes counter
    wire clr_h;            // Clear hours counter

    // Display registers
    reg [3:0] h_tens, h_ones;     // Hours display digits
    reg [3:0] m_tens, m_ones;     // Minutes display digits
    reg [3:0] s_tens, s_ones;     // Seconds display digits

    // Multiplexing registers
    reg [2:0] digit_select;       // Selects which digit to display (0-5)
    reg [19:0] refresh_counter;   // Counter for display refresh rate
    reg [3:0] current_digit;      // Current digit value to display

    // Main counter for 1-second timebase
    always @(posedge clk) begin
        if(rst || clr_cnt)
            count &lt;= 0;
        else
            count &lt;= count + 1;
    end

    // Seconds counter
    always @(posedge clk) begin
        if(rst || clr_s)
            scount &lt;= 0;
        else if(clr_cnt)
            scount &lt;= scount + 1;
```

```
    end

    // Minutes counter
    always @(posedge clk) begin
        if(rst || clr_m)
            mcount &lt;= 0;
        else if(clr_s)
            mcount &lt;= mcount + 1;
    end

    // Hours counter
    always @(posedge clk) begin
        if(rst || clr_h)
            hcount &lt;= 0;
        else if(clr_m)
            hcount &lt;= hcount + 1;
    end

    // Clear signals assignments
    assign clr_cnt = (count == 26'd49999999);  // 50MHz clock, one second
    assign clr_s = (scount == 6'd59) &amp;&amp; clr_cnt;
    assign clr_m = (mcount == 6'd59) &amp;&amp; clr_s;
    assign clr_h = (hcount == 5'd23) &amp;&amp; clr_m;

    // Convert binary counters to BCD for display
    always @(*) begin
        // Hours
        h_tens = hcount / 10;
        h_ones = hcount % 10;

        // Minutes
        m_tens = mcount / 10;
        m_ones = mcount % 10;

        // Seconds
        s_tens = scount / 10;
        s_ones = scount % 10;
    end

    // Refresh counter for multiplexing
    always @(posedge clk) begin
        if(rst)
            refresh_counter &lt;= 0;
        else
            refresh_counter &lt;= refresh_counter + 1;
    end

    // Digit selection for multiplexing (cycles through all 6 digits)
    always @(posedge clk) begin
        if(rst)
            digit_select &lt;= 0;
        else if(refresh_counter == 0)  // Change digit at a lower frequency
            digit_select &lt;= digit_select + 1;

        if(digit_select &gt; 5)
            digit_select &lt;= 0;
```

```
    end

    // Multiplexing logic - select which digit to display
    always @(*) begin
        case(digit_select)
            3'd0: current_digit = h_tens;
            3'd1: current_digit = h_ones;
            3'd2: current_digit = m_tens;
            3'd3: current_digit = m_ones;
            3'd4: current_digit = s_tens;
            3'd5: current_digit = s_ones;
            default: current_digit = 4'h0;
        endcase
    end

    // Command signals (anode control) for 6-digit display
    reg [5:0] cmd_reg;
    always @(*) begin
        cmd_reg = 6'b111111;  // All off (active low)
        cmd_reg[digit_select] = 1'b0;  // Turn on selected digit
    end
    assign Comnd = cmd_reg;

    // 7-segment decoder for the digits
    reg [6:0] segment_reg;
    always @(*) begin
        case(current_digit)
            4'd0: segment_reg = 7'b1000000;  // Display 0
            4'd1: segment_reg = 7'b1111001;  // Display 1
            4'd2: segment_reg = 7'b0100100;  // Display 2
            4'd3: segment_reg = 7'b0110000;  // Display 3
            4'd4: segment_reg = 7'b0011001;  // Display 4
            4'd5: segment_reg = 7'b0010010;  // Display 5
            4'd6: segment_reg = 7'b0000010;  // Display 6
            4'd7: segment_reg = 7'b1111000;  // Display 7
            4'd8: segment_reg = 7'b0000000;  // Display 8
            4'd9: segment_reg = 7'b0010000;  // Display 9
            default: segment_reg = 7'b1111111;  // All segments off
        endcase
    end
    assign Segment = segment_reg;

endmodule
```

Key Implementation Details

Counters:

- Added a `scount` register to count seconds from 0-59
- Modified the clear logic to create proper cascading between seconds, minutes, and hours

Digit Conversion:

- Added conversion from binary counters to BCD digits for each digit position

- Created h_tens, h_ones, m_tens, m_ones, s_tens, s_ones for the 6 display positions

Multiplexing Logic:

- Implemented a digit selection system that cycles through all 6 digits
- Used a refresh counter to control the display refresh rate
- Created command signals to control which digit is active at any given time

7-Segment Decoder:

- Added a BCD to 7-segment decoder based on standard patterns
- The segment values are active low (0 turns on a segment)

This implementation provides a complete 24-hour format real-time clock with all six digits (HH:MM:SS) displayed on a multiplexed 7-segment display. The multiplexing occurs fast enough that all digits appear to be continuously displayed to the human eye.

⁂