# Lab 4 - Stopwatch

Nathaniel Fargo

ECE 3700

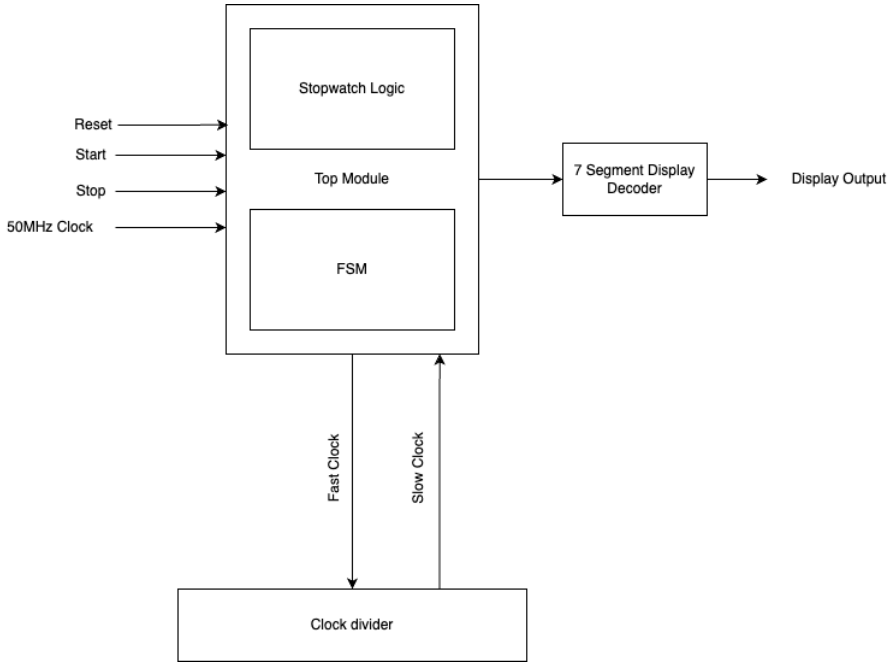
## 1. Introduction

This lab focused on designing and implementing a digital stopwatch on the DE10-Lite FPGA board using Verilog. The stopwatch counts from 0 to 9 in one-second intervals and can be controlled using start/resume, stop/pause, and reset inputs. The project aimed to enhance our understanding of sequential circuits, clock division, and finite state machine (FSM) design.

## 2. Overview of Design

The design comprises multiple Verilog modules, including a top-level module with an FSM, a clock divider, and a binary-to-7-segment display converter. The top module orchestrates the stopwatch's operation, responding to input signals and updating the display accordingly.

#### BLOCK DIAGRAM OF TOP-LEVEL MODULE



**TOP.V CODE**

```v
module Top (
    input clk_50mhz,
    input reset,
    input start,
    input stop,
    output [6:0] segment_display
);

    // hold the current output (0-9)
    reg [3:0] stopwatch_counter;

    // manage different states
    localparam RESET = 2'b00;
    localparam IDLE = 2'b01;
    localparam COUNTING = 2'b10;
    reg [1:0] state;

    initial begin
        stopwatch_counter <= 0;
        state <= IDLE;
    end

    // create clocks
    wire clk_1hz;
    wire clk_en = (state == COUNTING);

    ClkDiv50M clk_div_inst (
        .clk_50mhz(clk_50mhz),
        .clk_1hz(clk_1hz),
        .en(clk_en),
        .reset(reset)
    );

    // fast logic
    always @(posedge clk_50mhz) begin

        // handle inputs here
        if (reset) begin
            state <= RESET;
        end else if (~stop | (state == RESET)) begin
            state <= IDLE;
        end else if (~start) begin
            state <= COUNTING;
        end

    end

    // slow logic
    always @(posedge clk_1hz) begin
        // FSM
        case (state)
            IDLE: begin
                // do nothing
            end
            COUNTING: begin
                // increment stopwatch
                stopwatch_counter <= stopwatch_counter + 1;
                if (stopwatch_counter == 9)
                    stopwatch_counter <= 0;
            end
            RESET: begin
                // reset stopwatch
                stopwatch_counter <= 0;
            end
        endcase
    end

    // instatiate hex to 7 segment display
    HexTo7Seg segment_display_converter_inst (
        .hex_input(stopwatch_counter),
        .segment_display(segment_display)
    );

endmodule
```

Above is the code that implements the various submodules and interfaces with the FPGA pins.

# 3. Binary to 7-Segment Display

The HexTo7SegmentDisplay.v module converts a 4-bit BCD output into signals for the 7-segment display. This enables the stopwatch to visually display numbers from 0 to 9.

**HEXTO7SEGMENTDISPLAY.V CODE**

```v
/*
Uses the 7-segment display lights on the MAX10 to indicate the
number counted by four slide switches.
*/
module HexTo7Seg(
    input [3:0] hex_input,
    output reg [6:0] segment_display
);

    always@(*) begin
        // "always@(*)" (using a *) means "be sensitive to all the inputs".
        // If you only want to be sensitive to some inputs or variables then specify which ones rath
        // for example: "always@(hex_input[1])" will trigger all code in the always block any time t
        // The seven-segment lights on your board are "active low" (hense the use of the negation sy
        // See page 25 in the user manual to see what "active low" mean. The push buttons on your FP
        case (hex_input)
            4'h0: segment_display = ~7'b0111111; // 0
            4'h1: segment_display = ~7'b0000110; // 1
            4'h2: segment_display = ~7'b1011011; // 2
            4'h3: segment_display = ~7'b1001111; // 3
            4'h4: segment_display = ~7'b1100110; // 4
            4'h5: segment_display = ~7'b1101101; // 5
            4'h6: segment_display = ~7'b1111101; // 6
            4'h7: segment_display = ~7'b0000111; // 7
            4'h8: segment_display = ~7'b1111111; // 8
            4'h9: segment_display = ~7'b1101111; // 9
            4'hA: segment_display = ~7'b1110111; // A
            4'hB: segment_display = ~7'b1111100; // b
            4'hC: segment_display = ~7'b1011000; // c
            4'hD: segment_display = ~7'b1011110; // d
            4'hE: segment_display = ~7'b1111001; // E
            4'hF: segment_display = ~7'b1110001; // F
            default: segment_display = ~7'b0000000; // all lights off
        endcase
    end
endmodule
```

# 4. Clock Divider

The ClkDiv50M.v module divides the 50 MHz input clock from the DE10-Lite board to a 1 Hz output, providing a timing signal suitable for real-time counting.

This module uses a 32-bit counter to count clock cycles and toggles the output when it reaches 25 million. The divided clock signal ensures that the stopwatch increments at one-second intervals, aligning with human perception.

**CLKDIV50M.V CODE**

```
module ClkDiv50M (
    input clk_50mhz,
    output reg clk_1hz,
    input en,
    input reset
);

    reg [31:0] clk_div = 0;
    reg reset_prev = 0;

    always @(posedge clk_50mhz) begin

        reset_prev <= reset;

        if (reset) begin

            // reset counter
            clk_div <= 0;

            // quick rising edge on clock reset
            if (~reset_prev)
                clk_1hz <= 0;
            else
                clk_1hz <= 1;

        end else if (en) begin

            clk_div <= clk_div + 1;

            if (clk_div == 24_999_999) begin
            //if (clk_div == 500) begin
                clk_div <= 0;
                clk_1hz <= ~clk_1hz;
            end

        end

    end

endmodule
```

# 5. Simulation

The tb_Top.v module simulates the behavior of the Top.v module by applying various input signals to validate the stopwatch's functionality.

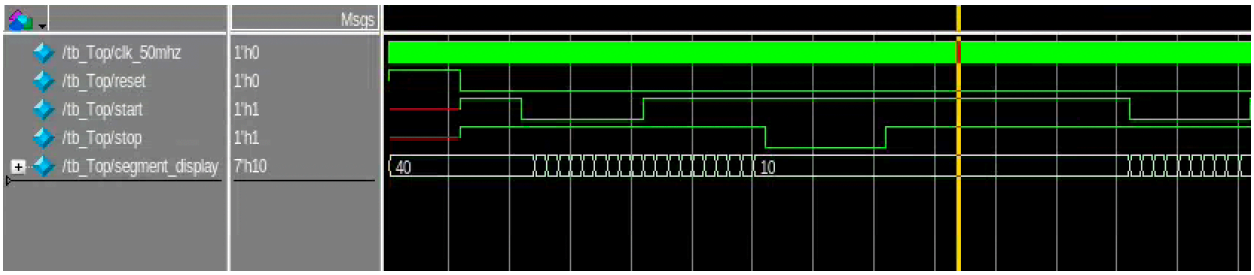
The testbench covers scenarios such as starting, pausing, and resetting the stopwatch. The simulation verified that the FSM transitioned correctly between states and that the counter behaved as expected.

**TB_TOP.V CODE**

```
module tb_Top;
   reg clk_50mhz;
   reg reset;
   reg start;
   reg stop;
   wire [6:0] segment_display;

   Top dut (
      .clk_50mhz(clk_50mhz),
      .reset(reset),
      .start(start),
      .stop(stop),
      .segment_display(segment_display)
   );

   initial begin
      clk_50mhz = 0;
      forever begin
         #10;
         clk_50mhz = ~clk_50mhz;
      end
   end

   initial begin

      reset = 1;
      #120000;
      reset = 0;
      start = 1;
      stop = 1;
      #100000;
      start = 0;
      #200000;
      start = 1;
      #200000;
      stop = 0;
      #200000;
      stop = 1;
      #300000;

      #100000;
      start = 0;
      #200000;
      start = 1;

      $done;
   end
```

In order to simulate these signals in shorter amounts of time, the delay period of the slow clock module was shortened from 50 million to 500, and the following waveforms were observed.

**SIMULATION WAVEFORM**

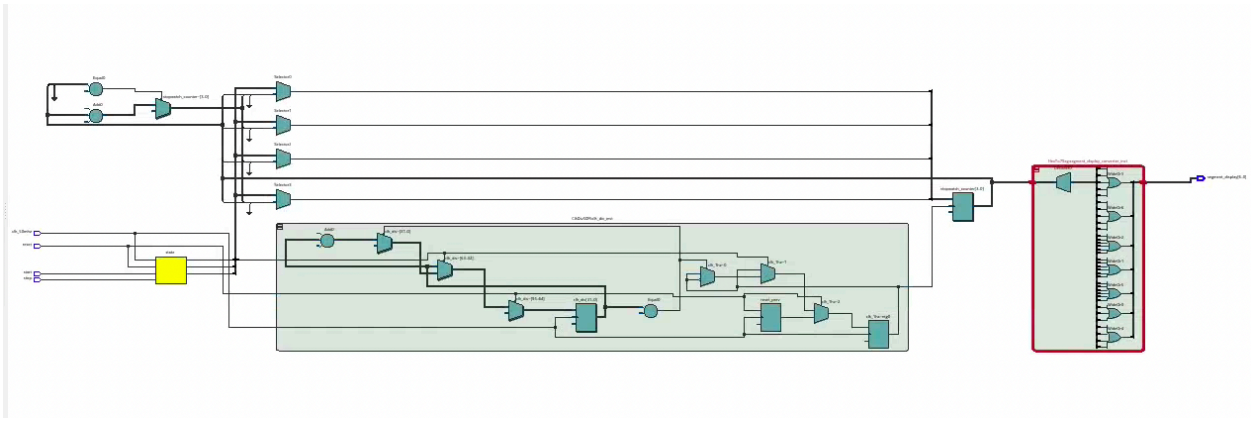



We can see the signals are changing once START is toggled, and stop once STOP is toggled. Additionally the reset functionality was proven to work as expected.

# 6. Netlist Analysis

For this lab we can also view the setlist output. Below is the generate RTL setlist.

**RTL NETLIST SCHEMATIC**



The synthesis report confirmed that the design was correctly mapped and placed, indicating that the FSM and sequential logic were effectively implemented. The netlist highlighted the clock divider's role in synchronizing the counting process with the reduced clock signal. We also see a couple of flip-flop/latch components that indicate the use of a clock in sequential logic.

# 7. Conclusion

This lab successfully demonstrated the design and implementation of a sequential circuit using Verilog. The completed stopwatch met all functional requirements, counting accurately in response to input signals. The lab reinforced the importance of modular design, simulation, and synthesis in digital circuit development.