CPEN-430 Lab Report

Course:	Digital System Design Lab
Lab Number & Title:	Lab 3: Verilog Procedural Combinational Coding Style
Student Name(s):	Gabriel DiMartino
Date:	September 11, 2025

Objective

The purpose of this lab was to design and implement a 2-bit comparator using a **procedural combinational Verilog coding style**. The circuit compares two unsigned 2-bit inputs, `A` and `B`, and produces three outputs:

- `AGTB` : A $>$ B
- `AEQB` : A $=$ B
- `ALTB` : A $<$ B

The results are displayed on LEDs as well as four 7-segment displays. Switches on the prototyping board serve as inputs. As an addition step, a synchronization module was added so that all switch inputs are sampled on the 50 MHz system clock, ensuring reliable operation.

Background / Theory

This design is based on **combinational logic**, where the output depends only on the current input values. (Memoryless)

- **Comparator**: Determines the relative magnitude of two binary numbers. For 2-bit unsigned numbers, there are three possible results ($A > B$, $A = B$, $A < B$).
- **Truth Table**: For each combination of A and B (16 cases total), exactly one of the three outputs should assert.
- **Procedural Coding Style**: The design was described using `always @(*)` blocks in Verilog.
- **Synchronization**: Switch inputs were sampled using a clocked synchronization process with the 50 MHz FPGA clock.

Materials and Equipment

- Altera DE2-115 FPGA prototyping board
- 4 slide switches (A[1:0], B[1:0]) as inputs
- 3 LEDs (`AGTB`, `AEQB`, `ALTB`)
- 4 seven-segment displays

- On-board 50 MHz oscillator
- Intel Quartus Prime (synthesis, implementation), ModelSim (simulation)
- Cyclone IV FPGA

Procedure

1. Wrote Verilog module for comparator using procedural combinational style.
2. Added synchronization registers clocked by the 50 MHz system clock.
3. Wrote testbench to exhaustively test all 16 input combinations.
4. Simulated design in ModelSim and verified waveforms.
5. Reviewed spec sheet for pin assignments.
6. Synthesized and implemented design in Quartus.
7. Reviewed synthesis schematic, timing, and area reports.

Results

- **Simulation**: Waveforms confirmed comparator output matched truth table for all 16 input cases.
- **Synthesized schematic**: Showed comparator built from LUTs and simple logic gates.
- **Hardware demonstration**: LEDs and 7-segment displays correctly showed results; synchronization eliminated glitches.

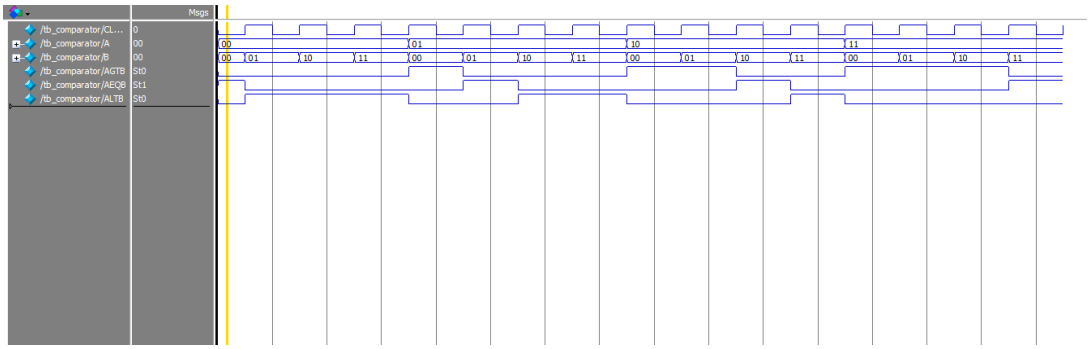

Figure 1: Simulation waveforms confirming comparator behavior.

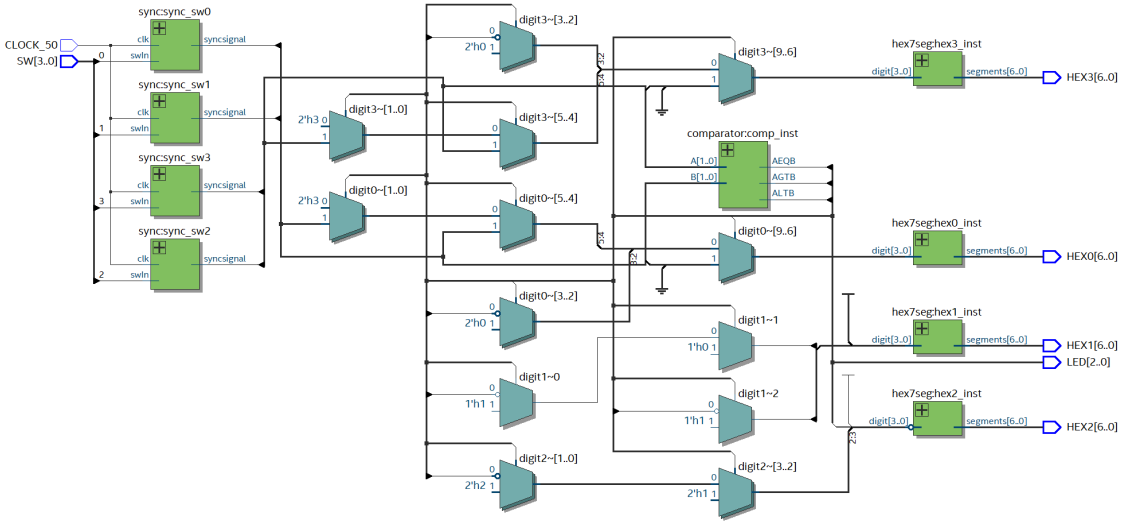

Figure 2: Synthesized schematic of the 2-bit comparator circuit with LED and 7 Segment display.

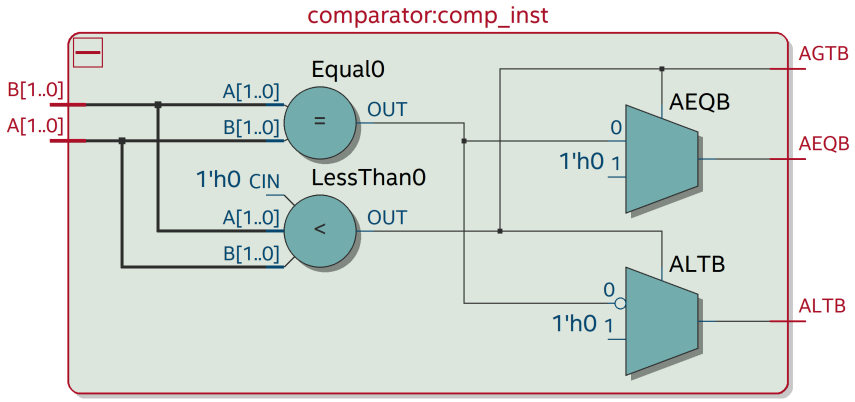

Figure 3: Synthesized schematic of the 2-bit comparator circuit.

Analysis

- Results matched predictions after synchronization was added.
- Synchronization logic cost negligible resources and eliminated hazards.
- Procedural coding style improved readability and modularity.

Conclusion

This lab successfully demonstrated the design, simulation, synthesis, and implementation of a 2-bit comparator circuit on an FPGA.

- **Objective achieved**: Comparator functioned correctly in all cases.
- **Broader connection**: Reinforced concepts of combinational design, best practices in Verilog, and hardware synchronization.
- **Skills gained**: Verilog coding, testbench development, waveform analysis, FPGA synthesis/implementation, hardware debugging.

Appendix

Comparator Verilog Code

```
module comparator (
    input  wire [1:0] A,         // 2-bit input A
    input  wire [1:0] B,         // 2-bit input B
    output reg  AGTB,            // A greater than B
    output reg  AEQB,            // A equal to B
    output reg  ALTB             // A less than B
);

    // Procedural combinational logic block
    always @(*) begin
        // Default outputs
        AGTB = 1'b0;
        AEQB = 1'b0;
        ALTB = 1'b0;

        // Compare A and B
        if (A > B) begin
            AGTB = 1'b1;
        end
        else if (A == B) begin
            AEQB = 1'b1;
        end
        else begin
            ALTB = 1'b1;
        end
    end

endmodule
```

Synchronization

```
module sync
(
    input swIn,
    input clk,
    output syncsignal
);
    reg flop1;
    reg flop2;

    always@(posedge clk)
    begin
```

```
        flop1 <= swIn;
        flop2 <= flop1;
    end

    assign syncsignal = flop2;
endmodule
```

7 Segment Display

```
module hex7seg (
    input  wire [3:0] digit,    // 4-bit input for digit/letter
       encoding
    output reg  [6:0] segments   // 7-segment output
);

    // Encoding for digits 0-9 and letters G, T, L, E, Q
    always @(*) begin
        case (digit)
            4'h0: segments = 7'b1000000; // 0
            4'h1: segments = 7'b1111001; // 1
            4'h2: segments = 7'b0100100; // 2
            4'h3: segments = 7'b0110000; // 3
            4'h4: segments = 7'b0011001; // 4
            4'h5: segments = 7'b0010010; // 5
            4'h6: segments = 7'b0000010; // 6
            4'h7: segments = 7'b1111000; // 7
            4'h8: segments = 7'b0000000; // 8
            4'h9: segments = 7'b0010000; // 9
            4'hA: segments = 7'b0000010; // G (looks like 6)
            4'hB: segments = 7'b0000111; // t (lowercase t)
            4'hC: segments = 7'b1000111; // L
            4'hD: segments = 7'b0000110; // E
            4'hE: segments = 7'b0011000; // q (closest to Q)
            4'hF: segments = 7'b1111111; // blank (all segments off)
            default: segments = 7'b1111111; // All segments off
        endcase
    end

endmodule
```

Comparator Top

```
module comparator_top (
    input  wire CLOCK_50,
    input  wire [3:0] SW,         // Switches: SW[3:2] = A, SW[1:0] = B
    output wire [2:0] LED,        // LEDs: LED[2]=AGTB, LED[1]=AEQB, LED
       [0]=ALTB
    output wire [6:0] HEX0,       // Seven segment display 0 (rightmost)
    output wire [6:0] HEX1,       // Seven segment display 1
    output wire [6:0] HEX2,       // Seven segment display 2
    output wire [6:0] HEX3        // Seven segment display 3 (leftmost)
);
    // Internal signals
    wire [3:0] SW_sync;           // Synchronized switch signals
```

```
    wire [1:0] A, B;
    wire AGTB, AEQB, ALTB;
    reg [3:0] digit3, digit2, digit1, digit0;

    // Instantiate synchronizers for each switch input
    sync sync_sw0 (
        .swIn(SW[0]),
        .clk(CLOCK_50),
        .syncsignal(SW_sync[0])
    );

    sync sync_sw1 (
        .swIn(SW[1]),
        .clk(CLOCK_50),
        .syncsignal(SW_sync[1])
    );

    sync sync_sw2 (
        .swIn(SW[2]),
        .clk(CLOCK_50),
        .syncsignal(SW_sync[2])
    );

    sync sync_sw3 (
        .swIn(SW[3]),
        .clk(CLOCK_50),
        .syncsignal(SW_sync[3])
    );

    // Assign synchronized inputs
    assign A = SW_sync[3:2];
    assign B = SW_sync[1:0];

    // Instantiate magnitude comparator
    comparator comp_inst (
        .A(A),
        .B(B),
        .AGTB(AGTB),
        .AEQB(AEQB),
        .ALTB(ALTB)
    );

    // Assign outputs to LEDs
    assign LED[2] = AGTB;
    assign LED[1] = AEQB;
    assign LED[0] = ALTB;

    // Display control logic using procedural combinational style
    always @(*) begin
        if (AGTB) begin
            // Display format: "AGtB" where A and B are the actual
               values
            digit3 = {2'b00, A};      // A value (0-3)
            digit2 = 4'hA;            // 'G'
            digit1 = 4'hB;            // 't'
            digit0 = {2'b00, B};      // B value (0-3)
        end
        else if (AEQB) begin
```

```
            // Display format: "AEqB" where A and B are the actual
               values
            digit3 = {2'b00, A};      // A value (0-3)
            digit2 = 4'hD;            // 'E'
            digit1 = 4'hE;            // 'q'
            digit0 = {2'b00, B};      // B value (0-3)
        end
        else if (ALTB) begin
            // Display format: "ALtB" where A and B are the actual
               values
            digit3 = {2'b00, A};      // A value (0-3)
            digit2 = 4'hC;            // 'L'
            digit1 = 4'hB;            // 't'
            digit0 = {2'b00, B};      // B value (0-3)
        end
        else begin
            // Error state - should never happen
            digit3 = 4'hF;            // blank
            digit2 = 4'hF;            // blank
            digit1 = 4'hF;            // blank
            digit0 = 4'hF;            // blank
        end
    end

    // Instantiate seven segment decoders for each display
    hex7seg hex3_inst (
        .digit(digit3),
        .segments(HEX3)
    );

    hex7seg hex2_inst (
        .digit(digit2),
        .segments(HEX2)
    );

    hex7seg hex1_inst (
        .digit(digit1),
        .segments(HEX1)
    );

    hex7seg hex0_inst (
        .digit(digit0),
        .segments(HEX0)
    );
endmodule
```

Testbench

```
`timescale 1ns / 1ps

module tb_comparator;

    // Clock signal for synchronized version
    reg CLOCK_50;

    // Testbench signals
```

```
    reg  [1:0] A, B;
    wire AGTB, AEQB, ALTB;

    // Clock generation (50 MHz)
    initial begin
        CLOCK_50 = 0;
        forever #10 CLOCK_50 = ~CLOCK_50;  // 50 MHz clock
    end

    // Instantiate UUT (basic comparator)
    comparator uut (
        .A(A),
        .B(B),
        .AGTB(AGTB),
        .AEQB(AEQB),
        .ALTB(ALTB)
    );

    integer i, j;

    initial begin
        // Create VCD file for waveform viewing
        $dumpfile("comparator_waveform.vcd");
        $dumpvars(0, tb_comparator);

        // Display header
        $display("Time\tA\tB\tAGTB\tAEQB\tALTB");
        $display("------------------------------");

        // Loop through all 16 combinations using nonblocking
           assignments
        for (i = 0; i < 4; i = i + 1) begin
            for (j = 0; j < 4; j = j + 1) begin
                A <= i[1:0];
                B <= j[1:0];

                // Wait one clock to see the change propagate
                @(posedge CLOCK_50);

                $display("%0t\t%0b\t%0b\t%b\t%b\t%b",
                         $time, A, B, AGTB, AEQB, ALTB);
            end
        end

        $display("All combinations tested.");
        $finish;
    end

endmodule
```