EAST WEST UNIVERSITY Department of Computer Science & Engineering

# Project Report: 4-bit Magnitude Comparator

**Course Title:** Digital Logic Design

**Course Code:** CSE345

**Section No:** 07

**Submitted To:**

Nishat Tasnim

Lecturer

Department of Computer Science and Engineering

**Submitted By:**

**Group-10**

| Student Name | Student ID |
|---|---|
| Md Saiful Islam | 2022-3-60-045 |
| Umme Mukaddisa | 2022-3-60-317 |
| Shanghita Naha Sristy | 2022-3-60-311 |
| Ayon Adhikary | 2022-3-60-137 |

## Table of Contents

## 1. Problem Statement

In modern digital systems, comparing the magnitude of binary numbers is a fundamental operation used in processors, sorting circuits, memory address checking, and control systems. The aim of this project is to design, implement, and analyze a **4-bit magnitude comparator**—a combinational logic circuit that compares two 4-bit unsigned binary numbers:

- **$A = A_3\ A_2\ A_1\ A_0$**
- **$B = B_3\ B_2\ B_1\ B_0$**

The comparator produces **three distinct output signals**:

- **$A > B$**: High (logic 1) if A is greater than B
- **$A = B$**: High (logic 1) if A is equal to B
- **$A < B$**: High (logic 1) if A is less than B

These outputs must be **mutually exclusive**, meaning only one can be high at any given time.

system design techniques. It serves as a building block for more advanced arithmetic and decision-making circuits used in computer architecture and embedded systems.

## 2. Project Objectives

- Accurate Comparison: The circuit must correctly compare any pair of 4-bit binary numbers, covering all 16 possible values (0000 to 1111).
- Efficient Logic Design: The implementation should minimize circuit complexity and propagation delay by using an optimal number of logic gates (AND, OR, NOT, etc.).
- Scalability: The design should be easily extendable to higher-bit comparators (e.g., 8-bit, 16-bit) through cascading or modular expansion.
- Platform Independence: The comparator should be implementable via:

- Gate-level design
- Simulation tools (e.g., Logisim, Multisim)
- Hardware description languages (e.g., VHDL or Verilog)
- Physical implementation using discrete logic ICs on breadboards

This project demonstrates the practical application of combinational logic and deepens understanding of binary number systems, logic minimization, and real-world digital design. It serves as a foundational element for more complex arithmetic and decision-making circuits in computer architecture and embedded systems.

## 3. Design Details

### 3.1. Input and Output Specification

**Inputs:**

- 4-bit binary number A: $A_3$ (MSB), $A_2$, $A_1$, $A_0$ (LSB)
- 4-bit binary number B: $B_3$ (MSB), $B_2$, $B_1$, $B_0$ (LSB)

**Outputs:**

- **A > B**: High when A is greater than B
- **A = B**: High when A is equal to B
- **A < B**: High when A is less than B

Only one output is high at any time.

## 3.2. Logical Approach

The comparison process follows **a top-down evaluation** from the **Most Significant Bit (MSB)** to the **Least Significant Bit (LSB)**. This ensures higher-weight bits are evaluated first, giving accurate magnitude comparison results. The entire logic is implemented **without XOR/XNOR gates**, instead using only **AND, OR, and NOT gates**, consistent with your actual circuit diagram.

### 3.2.1. Equality Condition:

Equality is verified bitwise using the following logic for each bit pair:

- $E_0$ = ($A_0$ AND $B_0$) OR (NOT $A_0$ AND NOT $B_0$)
- $E_1$ = ($A_1$ AND $B_1$) OR (NOT $A_1$ AND NOT $B_1$)
- $E_2$ = ($A_2$ AND $B_2$) OR (NOT $A_2$ AND NOT $B_2$)
- $E_3$ = ($A_3$ AND $B_3$) OR (NOT $A_3$ AND NOT $B_3$)

These expressions confirm that each individual bit pair is either both 0 or both 1.

The full equality condition across all 4 bits is:

A = B = $E_3$ AND $E_2$ AND $E_1$ AND $E_0$

### 3.2.2. Greater Than Condition:

$G_3$ = $A_3$ AND NOT $B_3$
The logic checks whether A is greater than B, starting from the MSB. The expressions are:

- $G_3$ = $A_3$ AND (NOT $B_3$)
- $G_2$ = $A_2$ AND (NOT $B_2$) AND $E_3$
- $G_1$ = $A_1$ AND (NOT $B_1$) AND $E_3$ AND $E_2$
- $G_0$ = $A_0$ AND (NOT $B_0$) AND $E_3$ AND $E_2$ AND $E_1$

These ensure that the higher-priority bits are equal before considering lower-order bits.

$A > B = G_3$ OR $G_2$ OR $G_1$ OR $G_0$

### 3.2.3. Less Than Condition:

This logic similarly checks from MSB to LSB for the "less than" condition:

- **$L_3$ = (NOT $A_3$) AND $B_3$**
- **$L_2$ = (NOT $A_2$) AND $B_2$ AND $E_3$**
- **$L_1$ = (NOT $A_1$) AND $B_1$ AND $E_3$ AND $E_2$**
- **$L_0$ = (NOT $A_0$) AND $B_0$ AND $E_3$ AND $E_2$ AND $E_1$**

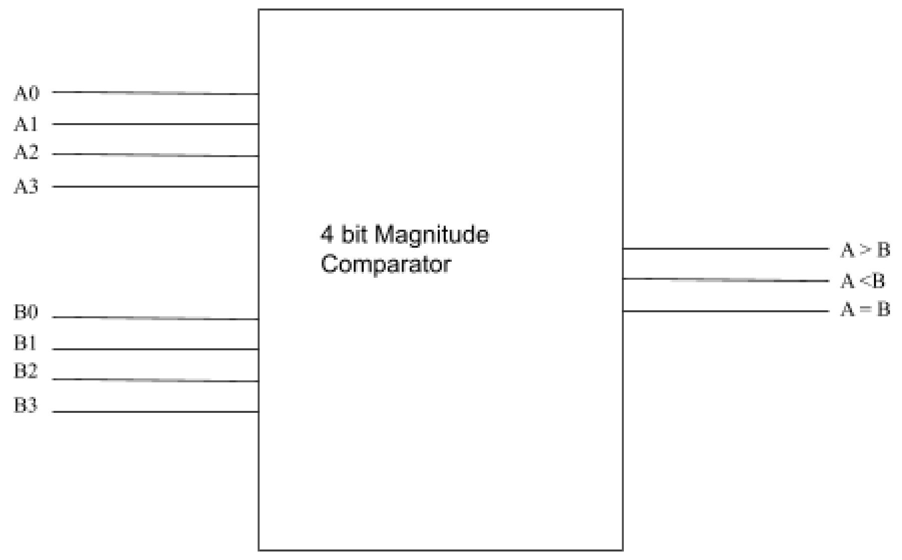
This guarantees that the less-than condition is only true if all more significant bits are equal and one lower bit in A is less than its B counterpart.

## 3.3. Block Diagram Description

The design consists of:

- Logic gates to verify bit-by-bit equality using AND, OR, and NOT
- Gate-level logic to evaluate "greater than" and "less than" comparisons
- A final logic layer that combines partial outputs into three mutually exclusive outputs:
  **A > B**, **A = B**, and **A < B**

Block Diagram:



## 4. Implementation Strategy

This comparator can be implemented in multiple ways:

- **Simulation Tools**: Designed and tested in tools like Logisim, Proteus, or Multisim
- **Hardware Implementation**: Built using discrete logic ICs (e.g., 7485 4-bit comparator chip)
- **HDL-Based Design**: Implemented in Verilog or VHDL and simulated on platforms like Vivado or EDA Playground

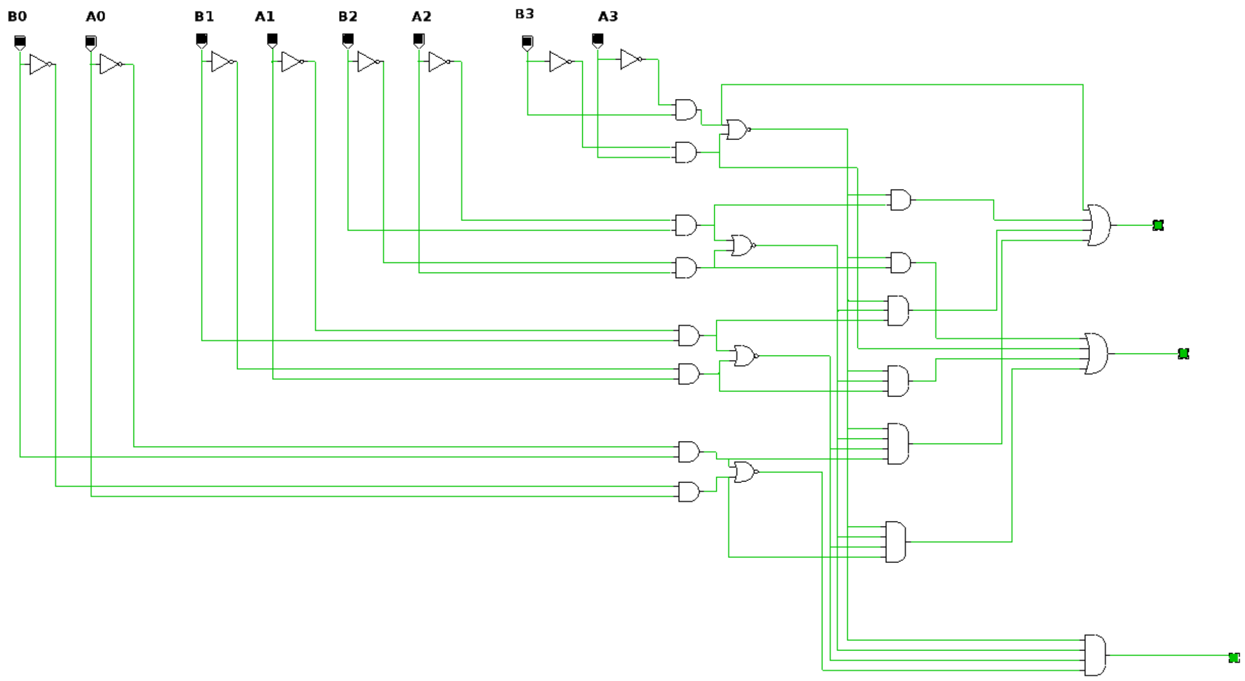
## 5. Advantages of the Design

- **Modular:** Easily expandable to 8-bit, 16-bit, or more
- **Reusable:** Can be embedded in ALUs and decision units in CPUs
- **Efficient:** Fully combinational and optimized for low delay and low power

## 6. Truth Table

| A3 A2 A1 A0 | B3 B2 B1 B0 | A > B | A = B | A < B |
|---|---|---|---|---|
| 0000 | 0000 | 0 | 1 | 0 |
| 0001 | 0000 | 1 | 0 | 0 |
| 0010 | 0011 | 0 | 0 | 1 |
| 0011 | 0011 | 0 | 1 | 0 |
| 0100 | 0011 | 1 | 0 | 0 |
| 0101 | 0101 | 0 | 1 | 0 |
| 0110 | 1000 | 0 | 0 | 1 |
| 0111 | 0110 | 1 | 0 | 0 |
| 1000 | 1000 | 0 | 1 | 0 |
| 1001 | 1010 | 0 | 0 | 1 |
| 1010 | 1001 | 1 | 0 | 0 |
| 1011 | 1011 | 0 | 1 | 0 |
| 1100 | 1110 | 0 | 0 | 1 |
| 1101 | 1100 | 1 | 0 | 0 |
| 1110 | 1110 | 0 | 1 | 0 |
| 1111 | 1110 | 1 | 0 | 0 |
| 1111 | 1111 | 0 | 1 | 0 |
| 0000 | 1111 | 0 | 0 | 1 |
| 1000 | 0111 | 1 | 0 | 0 |
| ….. | ….. | ….. | ….. | ….. |

## 7. Circuit Diagram



### Diagram Description

- Input: A3–A0 and B3–B0
- Output: A_gt_B, A_eq_B, A_lt_B
- Use XNOR gates for bitwise equality
- Use AND gates for bitwise combination
- Use OR gates to combine greater/less conditions

## 8. Behavioral Verilog Code and Simulation

### 8.1. Procedural Model

The procedural model uses the always block to describe the behavior of the 4-bit magnitude comparator. It evaluates the binary inputs A and B and determines whether A > B, A = B, or A < B.

## Design Code

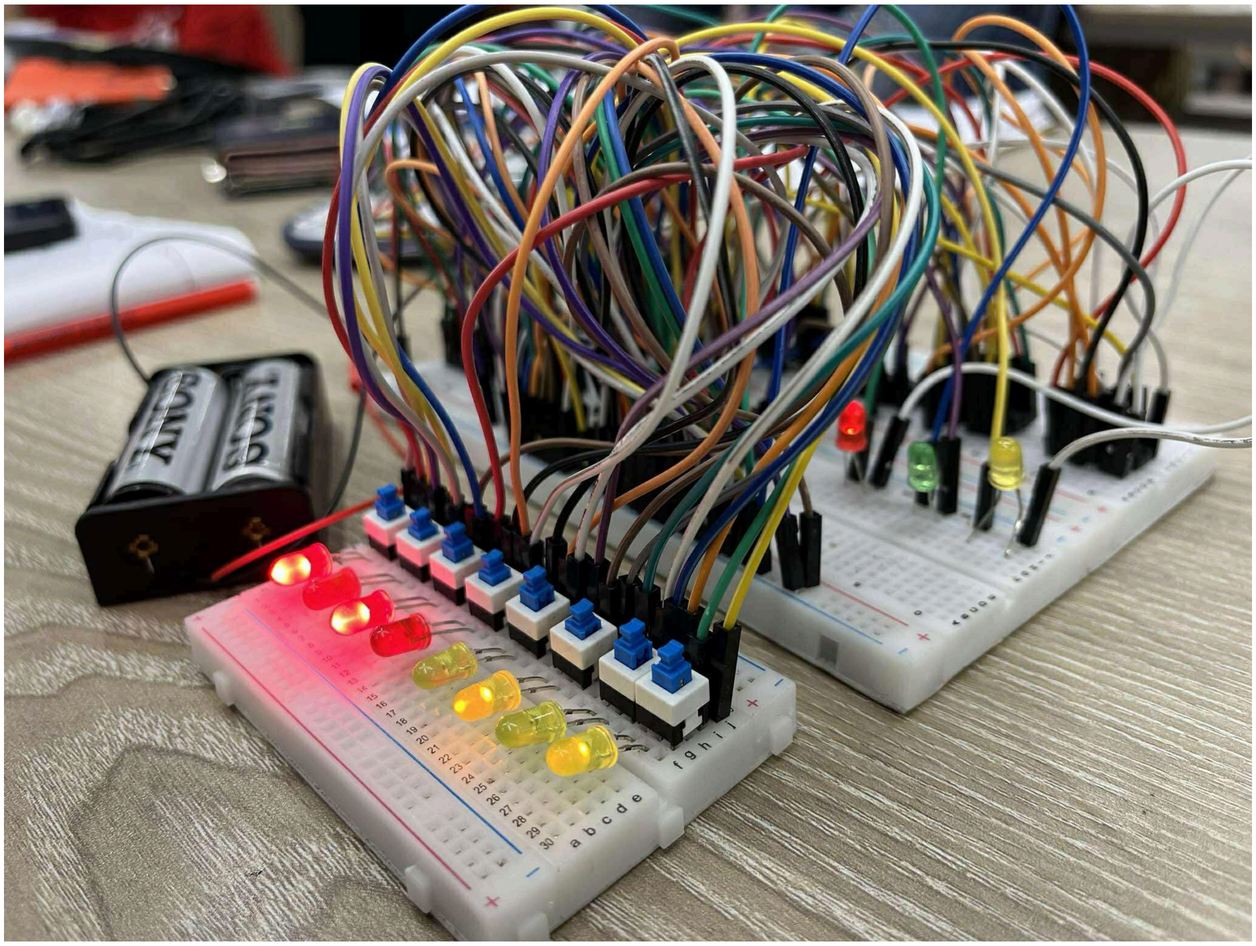
```
module comparator_4bit (
    input [3:0] A,
    input [3:0] B,
    output A_greater,
    output A_equal,
    output A_less
);
    wire E0, E1, E2, E3;

    // Bitwise equality (without XOR)
    assign E0 = (A[0] & B[0]) | (~A[0] & ~B[0]);
    assign E1 = (A[1] & B[1]) | (~A[1] & ~B[1]);
    assign E2 = (A[2] & B[2]) | (~A[2] & ~B[2]);
    assign E3 = (A[3] & B[3]) | (~A[3] & ~B[3]);

    assign A_equal = E3 & E2 & E1 & E0;

    wire G3, G2, G1, G0;
    assign G3 = A[3] & ~B[3];
    assign G2 = A[2] & ~B[2] & E3;
    assign G1 = A[1] & ~B[1] & E3 & E2;
    assign G0 = A[0] & ~B[0] & E3 & E2 & E1;

    assign A_greater = G3 | G2 | G1 | G0;

    wire L3, L2, L1, L0;
    assign L3 = ~A[3] & B[3];
    assign L2 = ~A[2] & B[2] & E3;
    assign L1 = ~A[1] & B[1] & E3 & E2;
    assign L0 = ~A[0] & B[0] & E3 & E2 & E1;

    assign A_less = L3 | L2 | L1 | L0;
endmodule
```

## TestBench

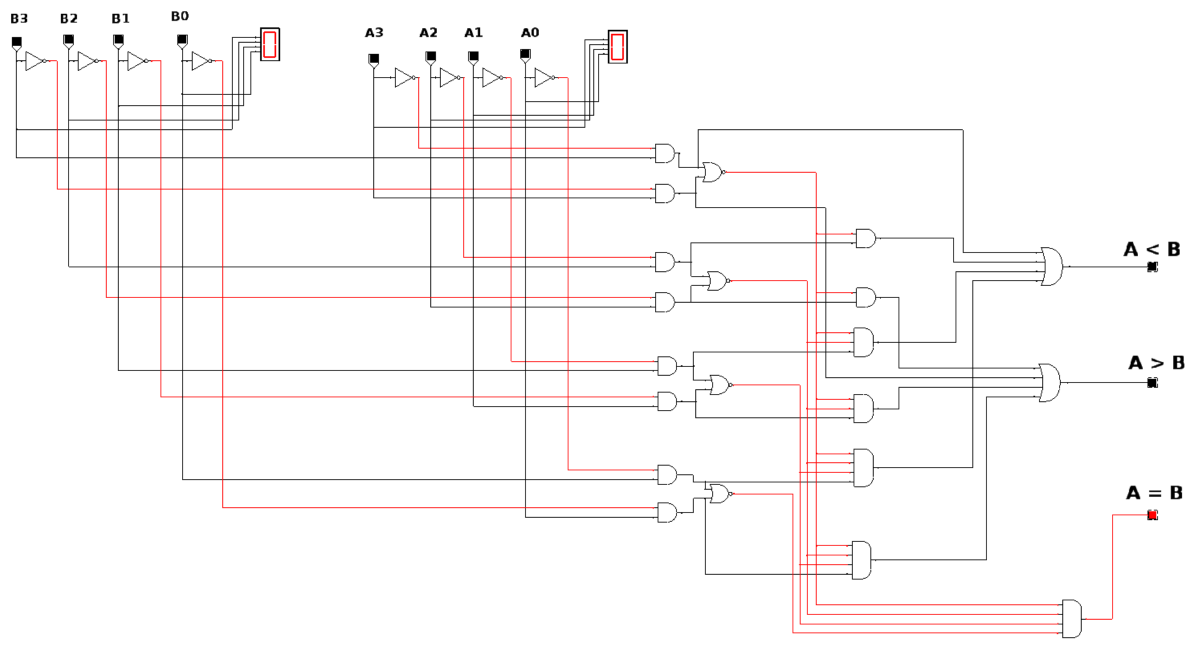
```
`timescale 1ns/1ps

module tb_comparator;

  reg [3:0] A, B;
  wire A_greater, A_equal, A_less;

  // Instantiate the comparator module
  comparator_4bit uut (
    .A(A),
    .B(B),
    .A_greater(A_greater),
    .A_equal(A_equal),
    .A_less(A_less)
  );

  // VCD file for waveform
  initial begin
    $dumpfile("dump.vcd");
    $dumpvars(0, tb_comparator);
  end

  // Apply stimulus
  initial begin
    A = 4'b0000; B = 4'b0000; #10;
    A = 4'b0101; B = 4'b0011; #10;
    A = 4'b1000; B = 4'b1000; #10;
    A = 4'b0010; B = 4'b1110; #10;
    A = 4'b1111; B = 4'b0001; #10;
    $finish;
  end
endmodule
```

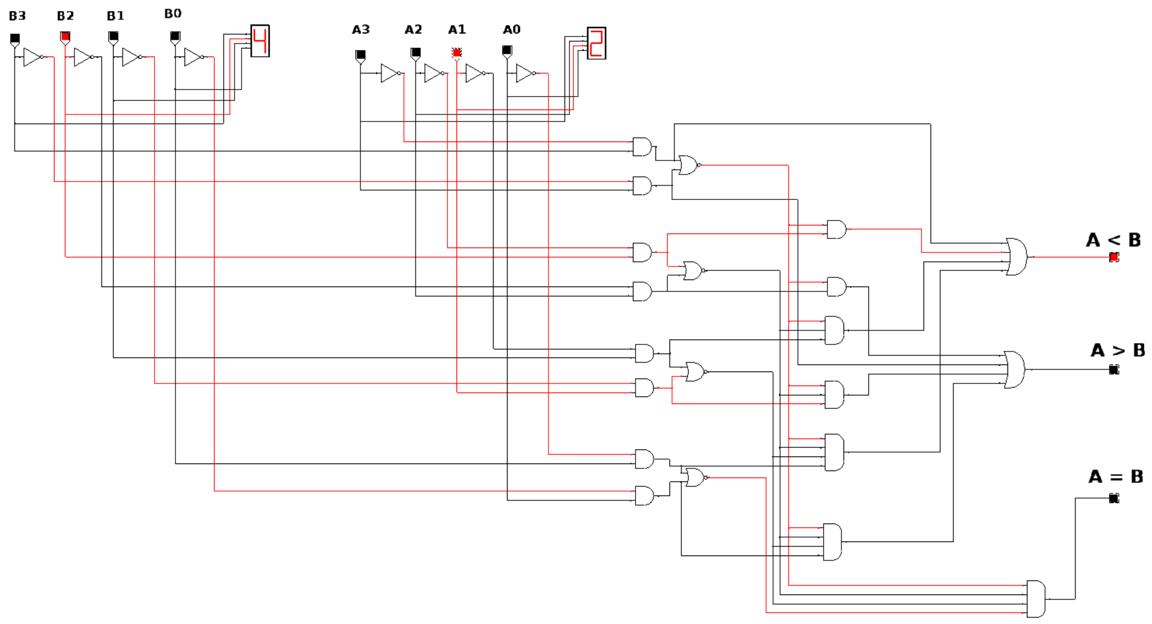
## 9. Simulation Results

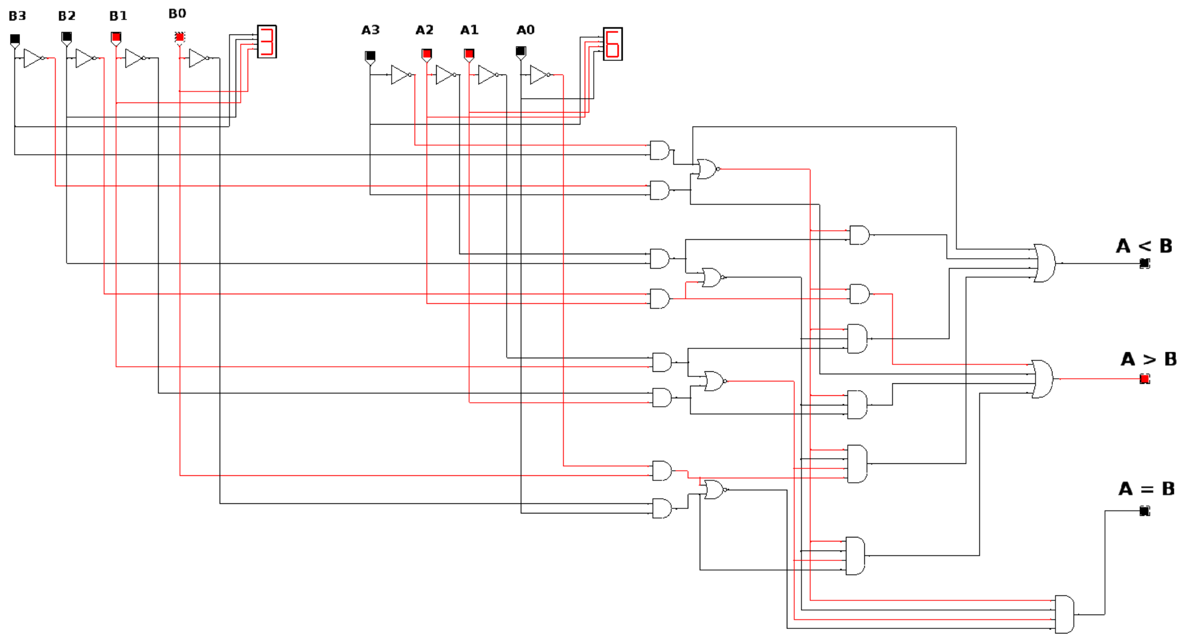
### 9.1. Screenshots



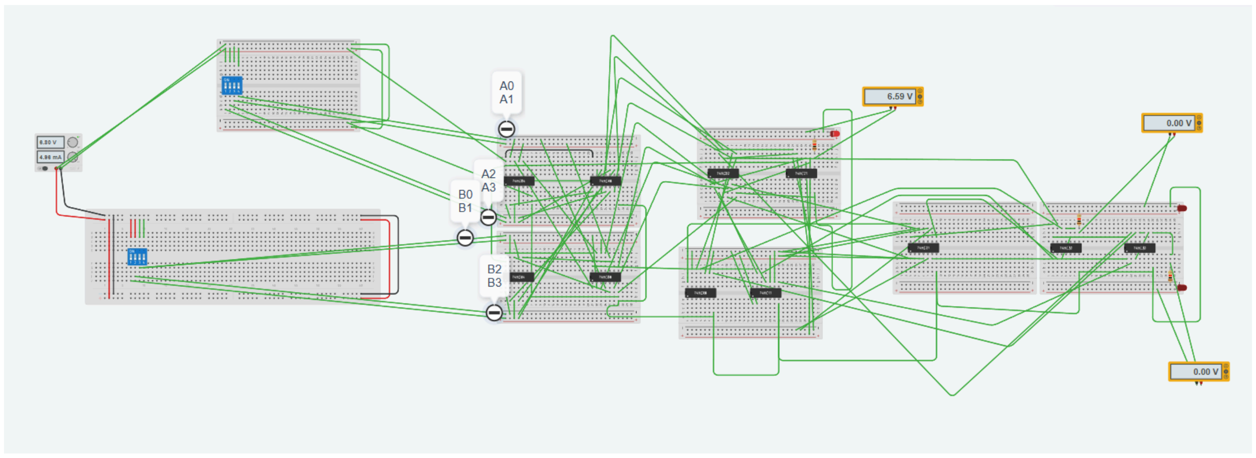
## 9.2. Logic Simulation





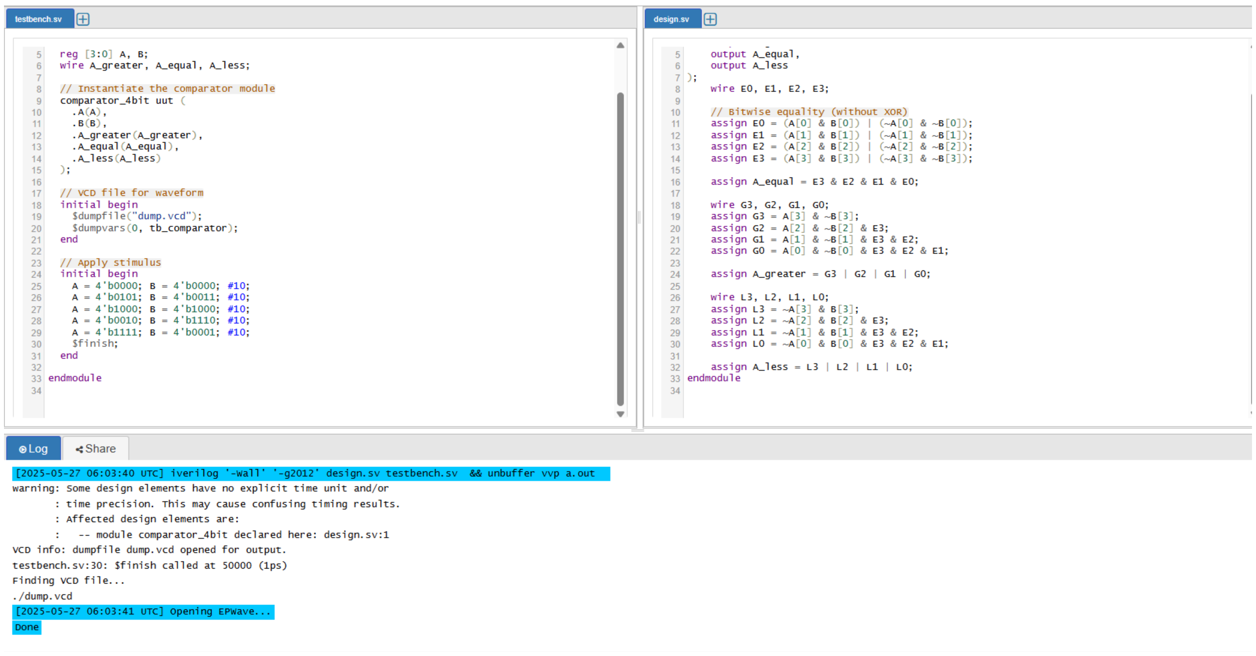

B3
B2
B1
B0
A3
A2
A1
A0
A < B
A > B
A = B

## TinkerCad:


A0
A1
A2
A3
B0
B1
B2
B3
6.59 V
0.00 V
0.00 V

## 9.3. Simulation Logs or Console

Here simulation results obtained using a testbench in EDA Playground. The testbench was designed to apply various binary values to A and B and monitor the comparator outputs.

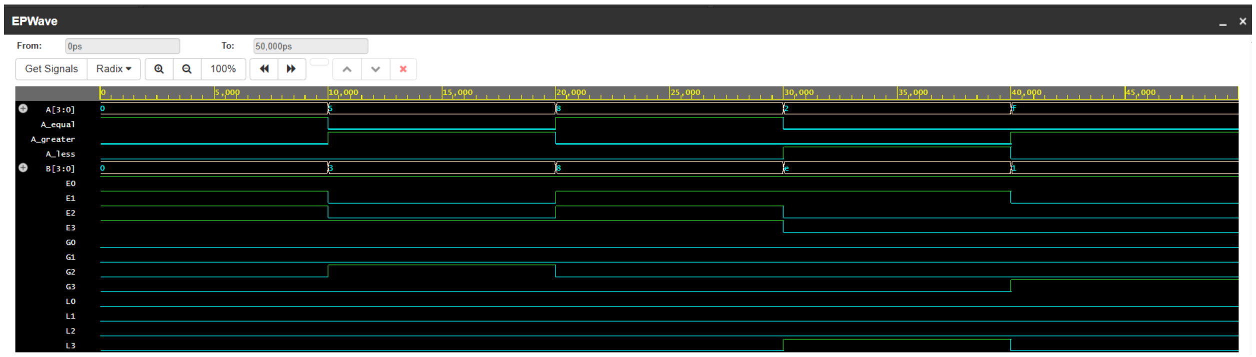


```verilog
 5    reg [3:0] A, B;
 6    wire A_greater, A_equal, A_less;
 7
 8    // Instantiate the comparator module
 9    comparator_4bit uut (
10      .A(A),
11      .B(B),
12      .A_greater(A_greater),
13      .A_equal(A_equal),
14      .A_less(A_less)
15    );
16
17    // VCD file for waveform
18    initial begin
19      $dumpfile("dump.vcd");
20      $dumpvars(0, tb_comparator);
21    end
22
23    // Apply stimulus
24    initial begin
25      A = 4'b0000; B = 4'b0000; #10;
26      A = 4'b0101; B = 4'b0011; #10;
27      A = 4'b1000; B = 4'b1000; #10;
28      A = 4'b0010; B = 4'b1110; #10;
29      A = 4'b1111; B = 4'b0001; #10;
30      $finish;
31    end
32
33  endmodule
34
```

```verilog
 5      output A_equal,
 6      output A_less
 7  );
 8      wire E0, E1, E2, E3;
 9
10      // Bitwise equality (without XOR)
11      assign E0 = (A[0] & B[0]) | (~A[0] & ~B[0]);
12      assign E1 = (A[1] & B[1]) | (~A[1] & ~B[1]);
13      assign E2 = (A[2] & B[2]) | (~A[2] & ~B[2]);
14      assign E3 = (A[3] & B[3]) | (~A[3] & ~B[3]);
15
16      assign A_equal = E3 & E2 & E1 & E0;
17
18      wire G3, G2, G1, G0;
19      assign G3 = A[3] & ~B[3];
20      assign G2 = A[2] & ~B[2] & E3;
21      assign G1 = A[1] & ~B[1] & E3 & E2;
22      assign G0 = A[0] & ~B[0] & E3 & E2 & E1;
23
24      assign A_greater = G3 | G2 | G1 | G0;
25
26      wire L3, L2, L1, L0;
27      assign L3 = ~A[3] & B[3];
28      assign L2 = ~A[2] & B[2] & E3;
29      assign L1 = ~A[1] & B[1] & E3 & E2;
30      assign L0 = ~A[0] & B[0] & E3 & E2 & E1;
31
32      assign A_less = L3 | L2 | L1 | L0;
33  endmodule
34
```

```
[2025-05-27 06:03:40 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer vvp a.out
warning: Some design elements have no explicit time unit and/or
       : time precision. This may cause confusing timing results.
       : Affected design elements are:
       :   -- module comparator_4bit declared here: design.sv:1
VCD info: dumpfile dump.vcd opened for output.
testbench.sv:30: $finish called at 50000 (1ps)
Finding VCD file...
./dump.vcd
[2025-05-27 06:03:41 UTC] Opening EPWave...
Done
```

## WaveForm:


EPWave
From: 0ps
To: 50,000ps
Get Signals
Radix
A[3:0]
A_equal
A_greater
A_less
B[3:0]
E0
E1
E2
E3
G0
G1
G2
G3
L0
L1
L2
L3