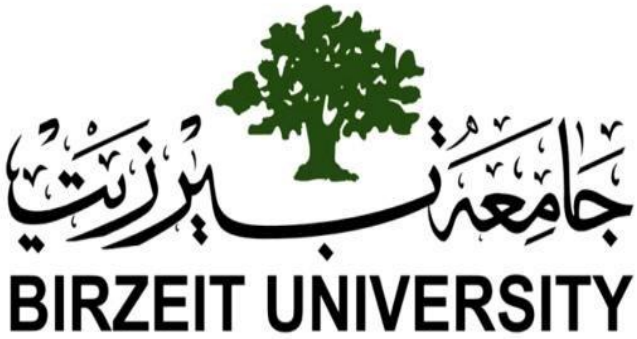

Faculty of Engineering & Technology
Electrical & Computer Engineering Department

ADVANCED DIGITAL SYSTEMS DESIGN– (ENCS3310)

Project Report

Prepared by:

Aws Hammad Id : 1221697

Instructor: Dr. Elias Khalil

Section: 2

Date: 26. December. 2024

Abstract

This project focuses on designing and implementing a structural comparator capable of handling 6-bit signed and unsigned numbers using basic logic gates. A select bit determines whether the inputs are treated as signed or unsigned. The comparator produces outputs indicating whether the first number is equal to, greater than, or less than the second. The design is constructed structurally using fundamental gates such as INV, NAND, NOR, AND, OR, XOR, and XNOR, with specified delays to assess maximum latency and determine the clock frequency. Additionally, the circuit incorporates error detection and operates synchronously with a clock signal. Its functionality is rigorously tested and validated through comprehensive simulations.

Table of contents

Table of Figures

Theory

1. Comparator:

The comparator is designed to evaluate two n-bit numbers and determine whether they are equal, greater than, or less than each other. It supports both signed numbers, represented using 2's complement for negative values, and unsigned numbers, where all values are non-negative. The implementation is structural, utilizing basic logic gates with integrated delays to simulate realistic timing behavior. A control signal, managed through a multiplexer, selects the mode of operation (signed or unsigned). This flexible design ensures the comparator is well-suited for accurate and dependable comparisons across a wide range of digital systems with varying bit widths [1].

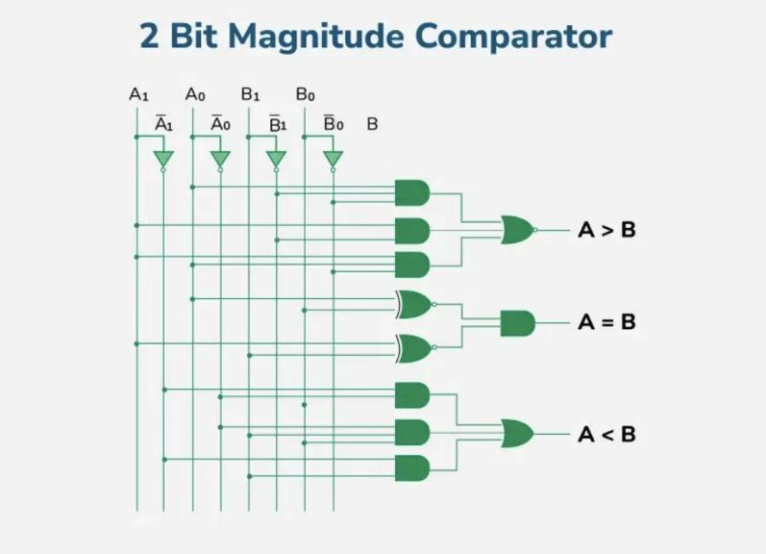

Figure 1: 2-Bit Comparator

2. 2×1 Multiplexer:

The 2×1 multiplexer, also called a 2-to-1 MUX, is a basic digital circuit used to select one of two input signals and forward it to the output. This MUX has two input lines, a single output line, and one selection line to control the choice of input. It finds numerous applications in digital systems, such as in microprocessors, where it is used to switch between two data sources or two different instructions[2].

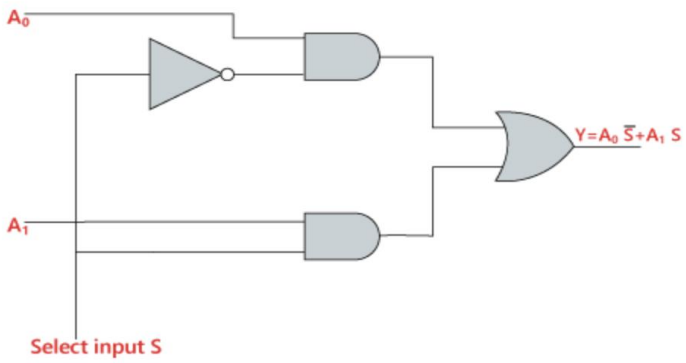

Figure 2: 2x1 Mux

3. Register:

The registers in the circuit are used to store the inputs A and B as well as the outputs Equal, Greater, and Smaller. These registers play a crucial role in synchronizing the comparator's operations with the clock signal. By updating their values only on the rising edge of the clock, the registers ensure that the circuit processes data in a stable and predictable manner. This synchronization helps maintain consistent operation in a sequential system, making the circuit both reliable and well-organized [3].

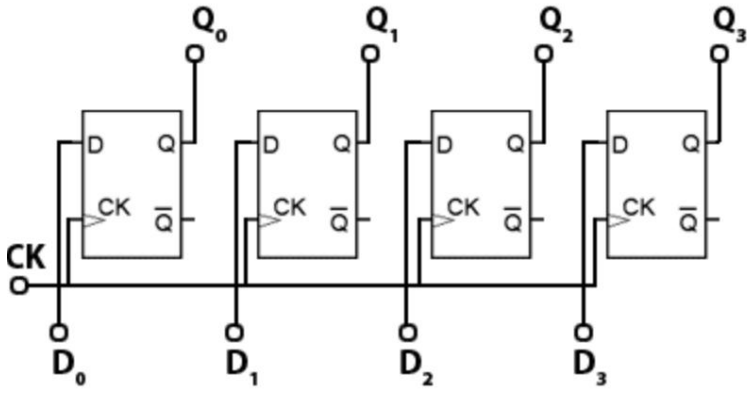

Figure 3: Register

Procedure & Data Analysis

1. Design and Implementation:

1.1 2-bits comparator module:

The 2-bits comparator module is designed to compare two 2-bits inputs **(a and b)** and generate outputs indicating whether a is greater than, equal to, or less than b. The module takes three additional inputs **(Gt_I, Eq_I, Lt_I)** to handle comparisons for multi-bit numbers, allowing it to cascade with other comparator stages. The internal logic uses basic gates to determine whether the most significant bit (MSB) or least significant bit (LSB) of a is greater than, equal to, or less than the corresponding bits of b. For equality, **XNOR** gates are used to compare the bits, while **AND** and **OR** gates combine these results to evaluate overall magnitude relationships. The module outputs **Gt_O, Eq_O, and Lt_O**, which represent whether a is greater than, equal to, or less than b, while accounting for cascading conditions from the previous stage. This modular design ensures accuracy and scalability for larger bit-width comparisons.

```
54  ////////////////////////////////////////////////////////////2-BIT-COMPARATOR////////////////////////////////////////////////////////////////////////////
55
56  module comparator2bits(a, b, Gt_I, Eq_I, Lt_I, Gt_O, Eq_O, Lt_O);
57      input [1:0] a, b; // 2-bit inputs to compare
58      input Gt_I, Eq_I, Lt_I; // inputs to compare multi-bit numbers
59      output Gt_O, Eq_O, Lt_O; // outputs of the comparison
60      wire not_a1, not_b1, not_a0, not_b0; // to have ~(inputs)
61      wire a1_and_not_b1, not_a1_and_b1; // to compare the MSB
62      wire a0_and_not_b0, not_a0_and_b0; // to compare the LSB
63      wire a1_eq_b1, a0_eq_b0; // if MSB or LSB are eqaul  for a,b
64      wire a_gt_b, a_lt_b, a_eq_b; // comparison results
65      wire a1_eq_b1_and_a0_gt_b0, a1_eq_b1_and_a0_lt_b0; // logic to help us in the comparison
66      wire a_eq_b_and_gt_i, a_eq_b_and_lt_i; // logic for multi-bit numbers
67      // to have the inverse of the inputs
68      not #2 (not_a1, a[1]);
69      not #2 (not_b1, b[1]);
70      not #2 (not_a0, a[0]);
71      not #2 (not_b0, b[0]);
72      // for greater and less comparisons (using AND gates with 6 delay)
73      and #6 (a1_and_not_b1, a[1], not_b1); // a[1] > b[1]
74      and #6 (not_a1_and_b1, not_a1, b[1]); // a[1] < b[1]
75      and #6 (a0_and_not_b0, a[0], not_b0); // a[0] > b[0]
76      and #6 (not_a0_and_b0, not_a0, b[0]); // a[0] < b[0]
77      // for equality comparisons  (using XNOR gates with 8 delay)
78      xnor #8 (a1_eq_b1, a[1], b[1]); // a[1] == b[1]
79      xnor #8 (a0_eq_b0, a[0], b[0]); // a[0] == b[0]
80      // greater logic (using OR gates with 6 delay)
81      and #6 (a1_eq_b1_and_a0_gt_b0, a1_eq_b1, a0_and_not_b0); // a[1:0] > b[1:0]
82      or #6 (a_gt_b, a1_and_not_b1, a1_eq_b1_and_a0_gt_b0); // a > b
83      // less logic
84      and #6 (a1_eq_b1_and_a0_lt_b0, a1_eq_b1, not_a0_and_b0); // a[1:0] < b[1:0]
85      or #6 (a_lt_b, not_a1_and_b1, a1_eq_b1_and_a0_lt_b0); // a < b
86      // equality logic
87      and #6 (a_eq_b, a1_eq_b1, a0_eq_b0); // a[1:0] == b[1:0]
88      // for multi-bits inputs
89      and #6 (a_eq_b_and_gt_i, a_eq_b, Gt_I); // equality and greater
90      and #6 (a_eq_b_and_lt_i, a_eq_b, Lt_I); // equality and less
91      // final results
92      or #6 (Gt_O, a_gt_b, a_eq_b_and_gt_i); // final greater output
93      or #6 (Lt_O, a_lt_b, a_eq_b_and_lt_i); // final less output
94      and #6 (Eq_O, a_eq_b, Eq_I); // final equality output
95  endmodule
```

Figure 4: 2-bits Comparator module

1.2 6-bits comparator module:

The 6-bit comparator is a hierarchical module that compares two 6-bit inputs (A and B) to determine if one is greater than, equal to, or smaller than the other. The comparator is designed to work in **both signed and unsigned** modes, selectable by the S input. Internally, the module uses three 2-bit comparators to divide and compare the inputs in stages: bits [1:0], [3:2], and [5:4]. The outputs of each stage are cascaded to compute the final comparison results for the full 6-bit inputs. For signed mode, the most significant bit (MSB), which acts as the sign bit, is handled separately to account for negative values. Multiplexers are used to select the correct comparison logic based on the MSB and the selector **(S)**. The results **(Equal, Greater, and Smaller)** are synchronized with the clock signal to ensure stable operation in a sequential system. This modular and synchronous design makes the comparator efficient, reliable, and versatile for different types of comparisons.

```
3  ////////////////////////////////////////////////////6-BIT-COMPARATOR//////////////////////////////////////////////////////////////////////
4  module comparator6bits_struct (clk,A,B,S,Equal,Greater,Smaller);
5       input wire clk,S;// the clock and the selecter between signed/unsigned
6       input wire [5:0] A,B;// 6-bits inputs A,B
7      output reg Equal,Greater,Smaller;// registers to have the outputs
8      reg [5:0] A_reg,B_reg;// registers to have inputs
9      reg S_reg;
10     wire Equal_signed, Greater_signed, Smaller_signed;// for the last signed result
11     wire Equal_unsigned, Greater_unsigned, Smaller_unsigned;
12     // put the inputs in registers to make the circuit synchronous
13     always @(posedge clk) begin
14         A_reg <= A;
15         B_reg <= B;
16         S_reg <= S;
17     end
18     wire gt_2, eq_2, lt_2; // for bits [1:0]
19     wire gt_4, eq_4, lt_4; // for bits [3:2]
20     wire gt_6, eq_6, lt_6; // for bits [5:4]
21     // compare bits [1:0]
22      comparator2bits c0(A_reg[1:0],B_reg[1:0],1'b0,1'b1,1'b0,gt_2,eq_2,lt_2);
23     // compare bits [3:2]
24      comparator2bits c1(A_reg[3:2],B_reg[3:2],gt_2,eq_2,lt_2,gt_4,eq_4,lt_4);
25     // compare bits [5:4]
26     comparator2bits c2(A_reg[5:4],B_reg[5:4],gt_4,eq_4,lt_4,gt_6,eq_6,lt_6);
27     wire msb_diff; // to see if the MSB of A and B are different
28     xor #9 (msb_diff,A_reg[5],B_reg[5]);
29      mux2x1 Equal_mux(.d0(eq_6), .d1(1'b0), .sel(msb_diff), .y(Equal_signed));// if MSBs are different select 1'b0 otherwise eq_6
30      mux2x1 Greater_msb_mux(.d0(gt_6), .d1(~A_reg[5]), .sel(msb_diff), .y(Greater_signed));// if MSBs different select ~A_reg[5] otherwise gt_6
31      mux2x1 Smaller_msb_mux(.d0(lt_6), .d1(A_reg[5]), .sel(msb_diff), .y(Smaller_signed));// if MSBs different select A_reg[5] otherwise lt_6
32     // unsigned compare
33     assign Equal_unsigned = eq_6;
34     assign Greater_unsigned = gt_6;
35     assign Smaller_unsigned = lt_6;
36     // to select between signed and unsigned results according to S
37     wire Equal_result, Greater_result, Smaller_result;
38     mux2x1 Equality_mux(Equal_unsigned, Equal_signed, S_reg, Equal_result);
39     mux2x1 Greater_mux(Greater_unsigned, Greater_signed, S_reg, Greater_result);
40     mux2x1 Smaller_mux(Smaller_unsigned, Smaller_signed, S_reg, Smaller_result);
41     // put the outputs in registers to make the circuit synchronous
42     always @(posedge clk) begin
43         Equal <= Equal_result;
44         Greater <= Greater_result;
45         Smaller <= Smaller_result;
46     end
47 endmodule
```

Figure 5: 6-bits Comparator module

1.3 Behavioral comparator module:

The behavioral 6-bit comparator uses high-level logic to compare two 6-bit inputs (A and B) in both signed and unsigned modes, determined by the selector input S. It operates by performing subtraction to compute the difference between the inputs. For unsigned mode, it calculates the subtraction using basic binary arithmetic and evaluates the **borrow** bit to determine if A is less than B. For signed mode, it interprets the inputs as signed numbers and uses **2's complement** arithmetic for subtraction. The results **(Equal, Greater, and Smaller)** are derived by checking the difference and are synchronized with the clock signal to ensure reliable operation in a sequential system. Additionally, the comparator stores the subtraction results for both signed and unsigned modes in separate registers, which can be useful for debugging or additional processing. This high-level implementation provides a clear and straightforward way to achieve accurate comparisons with less hardware complexity than a structural approach.

```
98  ////////////////////////////////////////////////////BEHAVIOURAL//////////////////////////////////////////////////////////////
99
100 module comparator6bits_behav (clk, A, B, S, Equal, Greater, Smaller, result_uns, result_sig);
101     input wire clk, S;// the clock and the selecter between signed/unsigned
102     input wire [5:0] A, B;// 6-bits inputs A,B
103     output reg Equal, Greater, Smaller;// registers to have the outputs
104     output reg [5:0] result_uns;// the result of the unsigned subtraction
105     output reg signed [5:0] result_sig;// the result of the signed subtraction
106     // registers to have inputs
107     reg [5:0] A_reg;
108     reg [5:0] B_reg;
109     reg S_reg;
110     wire [6:0] uns_sub;// for the unsigned result subtraction and the 7th bit is for the borrow
111     wire signed [5:0] sig_result;// for the signed result subtraction
112     wire signed [5:0] sig_A;// signed A
113     wire signed [5:0] sig_B;// signed B
114     // put the inputs in the registers to make the circuit synchronous
115     always @(posedge clk) begin
116         A_reg <= A;
117         B_reg <= B;
118         S_reg <= S;
119     end
120     // compute the unsigned and signed subtraction
121     assign uns_sub = {1'b0, A_reg} - {1'b0, B_reg};// the unsigned subtraction with borrow
122     assign sig_A = $signed(A_reg);// convert A to signed for signed subtraction
123     assign sig_B = $signed(B_reg);// convert B to signed for signed subtraction
124     assign sig_result = sig_A - sig_B;// signed subtraction
125     always @(posedge clk) begin
126         // put subtraction results in registers for signed and unsigned
127         result_uns <= uns_sub[5:0];
128         result_sig <= sig_result;
129         // compare by the selection
130         if (S_reg) begin // if s==1 then do signed comparison
131             Equal <= (sig_A == sig_B);// A == B
132             Greater <= (sig_A > sig_B);// A > B
133             Smaller <= (sig_A < sig_B);// A < B
134         end else begin
135             // if s==0 then do unsigned comparison
136             Equal <= (uns_sub[5:0] == 0);// A == B
137             Greater <= (!uns_sub[6]) && (uns_sub[5:0] != 0);// A > B
138             Smaller <= uns_sub[6];// A < B
139         end
140     end
141 endmodule
```

Figure 6: Behavioral comparator module

1.4 TestBench module:

The testbench is designed to thoroughly verify the functionality of both the **structural** and **behavioral** 6-bit comparators. It generates all possible input combinations for the 6-bit inputs A and B, cycling through **all 64 possible values** for each input. For each combination, the testbench tests both unsigned (S=0) and signed (S=1) modes. A clock signal is generated and toggled every 16 time units to synchronize the comparators. The task testing is used to compare the outputs of the behavioral and structural implementations, displaying the results for equality, greater-than, and smaller-than comparisons. If a mismatch is detected between the outputs of the two comparators, it flags the test as a failure; otherwise, it indicates a successful match. The testbench ensures comprehensive testing by exhaustively evaluating all input scenarios, verifying that the comparators function correctly in both signed and unsigned modes. This systematic approach guarantees the reliability and accuracy of the design.

```
144 ////////////////////////////////////////////////////////////////TESTBENCH////////////////////////////////////////////////////////////////////////////////////////////////////
145
146 module TB;
147     // inputs for the testbench
148     reg clk; // clock
149     reg [5:0] A; // 6-bit input A
150     reg [5:0] B; // 6-bit input B
151     reg S; // selector to select signed or unsigned
152     // outputs
153     wire behav_eq;// equality for behavioral
154     wire behav_gr;// greater for behavioral
155     wire behav_sm;// smaller for behavioral
156     wire [5:0] behav_uns;// unsigned subtraction result for behavioral
157     wire signed [5:0] behav_si;// signed subtraction result for behavioral
158     wire str_eq;// equality for structural
159     wire str_gr;// greater for structural
160     wire str_sm;// smaller for structural
161     // instance of the behavioral module
162     comparator6bits_behav c1 (clk, A, B, S, behav_eq, behav_gr, behav_sm, behav_uns, behav_si);
163     // instance the structural module
164     comparator6bits_struct c2 (clk, A, B, S, str_eq, str_gr, str_sm);
165     initial begin
166         clk = 0;// make the clock = 0
167         forever #16 clk = ~clk;// invert clock every 16 time units
168     end
169     // task to test the modules
170     task testing;
171         input [5:0] A;// input A for testing
172         input [5:0] B;// input B for testing
173         input S;// for signed or unsigned mode
174         begin
175             $display("Testing A=%b, B=%b, S=%b", A, B, S);// display inputs
176             $display("Behavioral -> Equal=%b, Greater=%b, Smaller=%b, Unsigned Result=%b, Signed Result=%d",
177                     behav_eq, behav_gr, behav_sm, behav_uns, behav_si);// display behavioral results
178             $display("Structural -> Equal=%b, Greater=%b, Smaller=%b", str_eq, str_gr, str_sm);// display structural results
179             if (behav_eq !== str_eq || behav_gr !== str_gr || behav_sm !== str_sm) begin
180                 $display("FAIL : Mismatch detected!");// display FAIL if outputs does not equal
181             end else begin
182                 $display("PASS : Behavioral and Structural outputs are identical.");// display PASS if outputs are equal
183             end
184             $display("------------------------");
185         end
186     endtask
187
```

```
188     // testing
189     integer i, j;// for the loop
190     initial begin
191         #32; // wait for first reset
192         // loop through ALL possible values for A and B
193         for (i = 0; i < 64; i = i + 1) begin // 2^6 = 64 possible value for each A and B
194             for (j = 0; j < 64; j = j + 1) begin
195                 A = i[5:0]; // give a 6-bit value to A
196                 B = j[5:0]; // give a 6-bit value to B
197                 // test for unsigned
198                 S = 0; // set to unsigned
199                 #128 testing(A, B, S); // call the testing task
200                 // test for signed
201                 S = 1; // set to signed
202                 #128 testing(A, B, S); // call the testing task
203             end
204         end
205         $stop; // stop simulation
206     end
207 endmodule
208
```

Figure 7: TestBench module

2. Testing and Verification:

The waveform of the testbench displays the clock signal (**clk**) and test inputs **A, B, and S** being fed to the comparator circuit under various conditions. The A and B signals represent the binary inputs to the comparator, while S selects the mode (unsigned or signed comparison). The outputs of both behavioral (**behav_uns** for unsigned and **behav_si** for signed) and structural implementations **(str_gr, str_eq, str_sm)** are displayed. The behavioral outputs match the structural outputs for all test cases, indicating the functionality is consistent across implementations.

The output confirms the synchronization between behavioral and structural comparators. Each test case verifies specific scenarios where A is either greater than, less than, or equal to B. The consistent "**PASS**" messages ensure the outputs for equal, greater, and smaller indicators (E, G, and L) are identical between the two implementations. This confirms the comparator's correctness and reliability for both signed and unsigned comparisons.

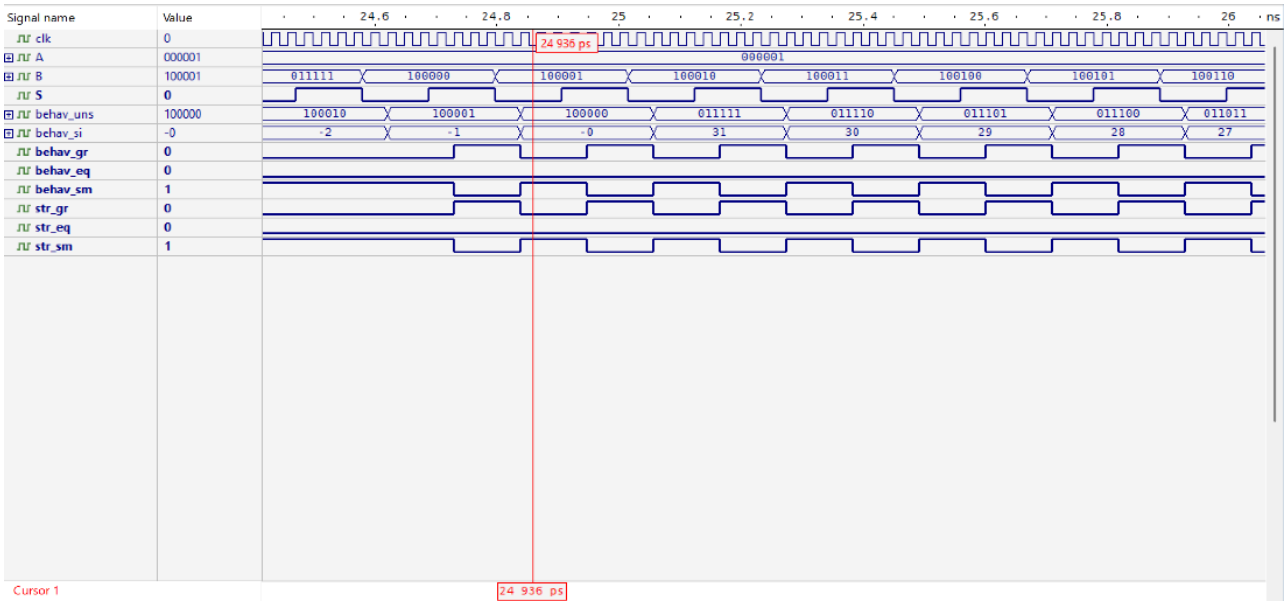

Figure 8: Waveform of the TestBench

```
run
# KERNEL: Testing A=111111, B=000110, S=1
# KERNEL: Behavioral -> Equal=0, Greater=0, Smaller=1
# KERNEL: Structural -> Equal=0, Greater=0, Smaller=1
# KERNEL: PASS : Behavioral and Structural outputs are identical :)
# KERNEL: ------------------------
# KERNEL: Testing A=111111, B=000110, S=0
# KERNEL: Behavioral -> Equal=0, Greater=1, Smaller=0
# KERNEL: Structural -> Equal=0, Greater=1, Smaller=0
# KERNEL: PASS : Behavioral and Structural outputs are identical :)
# KERNEL: ------------------------
# KERNEL: Testing A=011111, B=100000, S=1
# KERNEL: Behavioral -> Equal=0, Greater=1, Smaller=0
# KERNEL: Structural -> Equal=0, Greater=1, Smaller=0
# KERNEL: PASS : Behavioral and Structural outputs are identical :)
# KERNEL: ------------------------
# KERNEL: Testing A=011111, B=100000, S=0
# KERNEL: Behavioral -> Equal=0, Greater=0, Smaller=1
# KERNEL: Structural -> Equal=0, Greater=0, Smaller=1
# KERNEL: PASS : Behavioral and Structural outputs are identical :)
# KERNEL: ------------------------
# KERNEL: Testing A=010101, B=010101, S=1
# KERNEL: Behavioral -> Equal=1, Greater=0, Smaller=0
# KERNEL: Structural -> Equal=1, Greater=0, Smaller=0
# KERNEL: PASS : Behavioral and Structural outputs are identical :)
# KERNEL: ------------------------
# KERNEL: Testing A=010101, B=010101, S=0
# KERNEL: Behavioral -> Equal=1, Greater=0, Smaller=0
# KERNEL: Structural -> Equal=1, Greater=0, Smaller=0
# KERNEL: PASS : Behavioral and Structural outputs are identical :)
# KERNEL: ------------------------
# KERNEL: Testing A=111111, B=111111, S=1
# KERNEL: Behavioral -> Equal=1, Greater=0, Smaller=0
# KERNEL: Structural -> Equal=1, Greater=0, Smaller=0
# KERNEL: PASS : Behavioral and Structural outputs are identical :)
# KERNEL: ------------------------
# KERNEL: Testing A=111111, B=111111, S=0
# KERNEL: Behavioral -> Equal=1, Greater=0, Smaller=0
# KERNEL: Structural -> Equal=1, Greater=0, Smaller=0
# KERNEL: PASS : Behavioral and Structural outputs are identical :)
# KERNEL: ------------------------
# KERNEL: Testing A=000000, B=000000, S=1
# KERNEL: Behavioral -> Equal=1, Greater=0, Smaller=0
# KERNEL: Structural -> Equal=1, Greater=0, Smaller=0
# KERNEL: PASS : Behavioral and Structural outputs are identical :)
# KERNEL: ------------------------
```

Figure 9: Output of the TestBench

3. Maximum latency and Maximum frequency:

The maximum latency of the 6-bit comparator system can be calculated by adding the gate delays for each component in the design. The 6-bit comparator consists of three cascaded 2-bit comparators and two 2-to-1 multiplexers. Each 2-bit comparator introduces a delay of 26 units, accounting for 2 AND gates (6 units each), an XNOR gate (8 units), and an OR gate (6 units). With three 2-bit comparators, the total delay from these components is $3 \times 26 = 78$ units. Additionally, each 2-to-1 multiplexer adds a delay of 14 units, which includes 2 AND gates (6 units each), an OR gate (6 units), and a NOT gate (2 units), leading to a total delay of $2 \times 14 = 28$ units from the multiplexers. Therefore, the total maximum latency is $78 + 28 = 106$ units. Using the formula for the maximum clock frequency, which is the reciprocal of twice the maximum latency, the maximum frequency that can be applied to the registers is approximately $1 / (2 \times 106) = 4.72$ MHz.

The maximum latency in a digital system should be the minimum clock time that ensures correct operation and prevents any incorrect outputs. It represents the longest time it takes for the system to process and propagate signals through all components, ensuring that each stage of the system has enough time to complete its task before the next clock cycle begins. In the case of the 6-bit comparator system, the maximum latency is carefully calculated to be the least time required for the circuit to function properly without producing any erroneous results. Interestingly, despite the propagation delays, the system does not give a wrong output at any clock time, as the design ensures that all critical paths have sufficient time to settle before the next edge of the clock is applied. This guarantees that the outputs remain valid throughout the operation of the circuit.

4. Block Diagram of the design:

The block diagram represents a multi-bit comparator design that compares two binary numbers, A and B, and determines their relationship in terms of "greater than," "equal to," and "less than," supporting both signed and unsigned comparisons, controlled by a selection signal (S). The design utilizes **three 2-bit comparators** as building blocks, where each comparator processes two bits of A and B at a time, outputting local comparison results: A > B, A = B, and A < B. These outputs are cascaded to compute the **final 6-bit** comparison results. For unsigned comparisons, the results are directly derived from the individual two-bit comparators without considering sign bits. For signed comparisons, the sign bit (**MSB** of A and B) is taken into account, and logic gates along with multiplexers are used to handle the signed comparison. If S = 1, the sign bit is included in the comparison, while if S = 0, the comparison is based on unsigned logic. The multiplexers select between the signed and unsigned results based on the value of the S signal. The final outputs indicate whether A is greater than B, equal to B, or less than B, computed for both signed and unsigned operations.

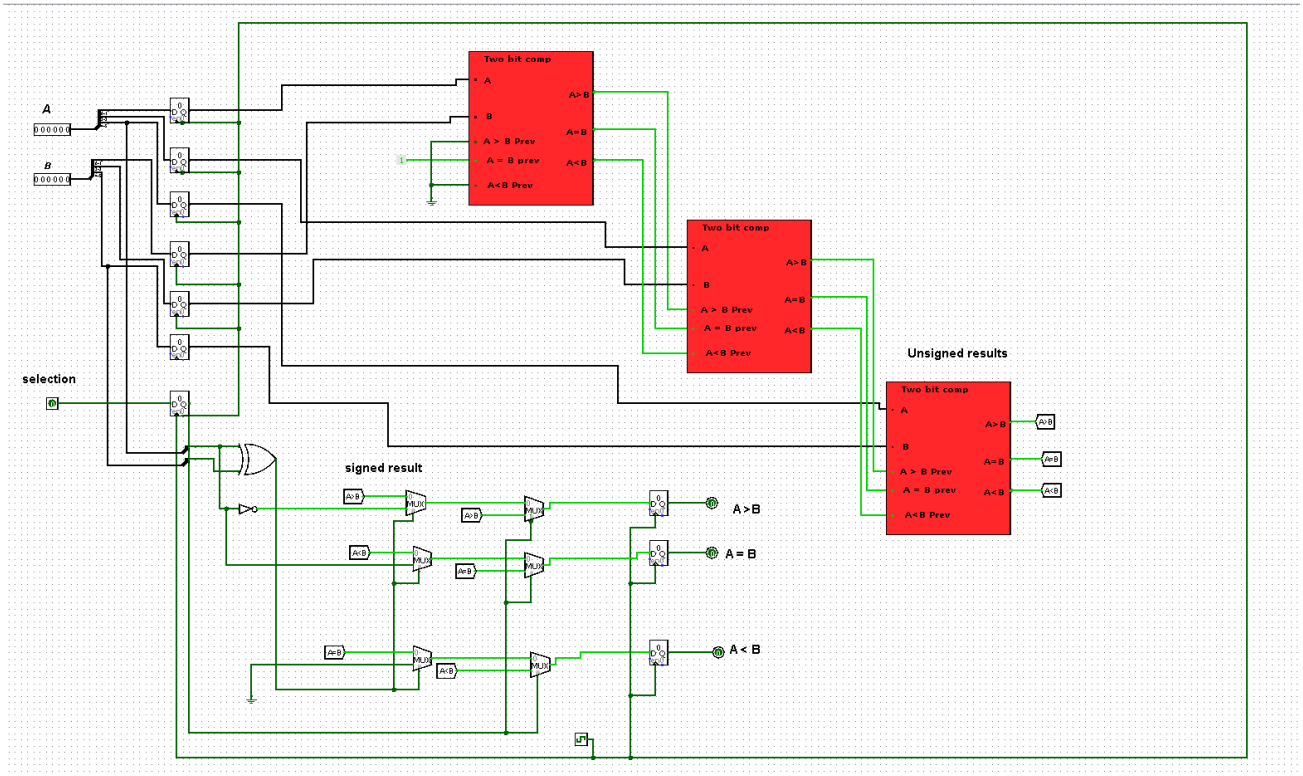

Figure 10: Block Diagram of the design

Conclusion

The project successfully developed and validated a 6-bit comparator that can handle both signed and unsigned numbers. The design was implemented using basic logic gates in a structural approach, ensuring modularity, clarity, and synchronization through the use of registers. A key element in the verification process was the testbench, which generated random test cases and compared the outputs from the structural design with those from a behavioral model. Errors were intentionally introduced to test the robustness of the testbench, confirming its ability to detect discrepancies. Simulation results demonstrated the comparator's accuracy across all valid input combinations. This project lays a solid foundation for future enhancements, including reducing delays, expanding the bit-width, and incorporating more advanced error-detection techniques.

References

[1] https://www.geeksforgeeks.org/magnitude-comparator-in-digital-logic/ **[Accessed 26-12-2024]**

[2] https://www.javatpoint.com/multiplexer-digital-electronics **[Accessed 27-12-2024]**

[3] https://learnabout-electronics.org/Digital/dig57.php **[Accessed 27-12-2024]**