Faculty of Engineering and Technology

Electrical and Computer Engineering Department

Advanced Digital Design ENCS3310

First Semester 2024 / 2025

Project

Student Name: Fadi Bassous **University ID: 1221005**

Instructor's name: Dr. Elias Khalil

Section: 2

Date: 27 /12 /2024

Abstract

This project involves the design and implementation of a structural comparator for 6-bit signed and unsigned numbers using basic logic gates. The comparator identifies whether the inputs are signed or unsigned via a select bit and provides outputs indicating whether the first number is equal to, greater than, or less than the second number. The design is implemented structurally using gates such as INV, NAND, NOR, AND, OR, XOR, and XNOR, with defined delays to evaluate maximum latency and clock frequency. The comparator circuit includes error detection and operates synchronously with a clock signal, validated through extensive simulations.

Table of contents

Table of Figures

List of Tables:

Theory and design

This project aims to design a structural comparator capable of processing signed and unsigned numbers represented in 6-bit binary format. The comparator is designed to provide accurate and reliable comparisons for three conditions: equality, greater than, and less than. Registers, implemented using D flip-flops, are used to synchronize the inputs and outputs with a clock signal, ensuring stability and preventing race conditions. By integrating error detection and synchronous operation, the project demonstrates the principles of advanced digital design and verification.(Figure 1 shows the full circuit block of the project)

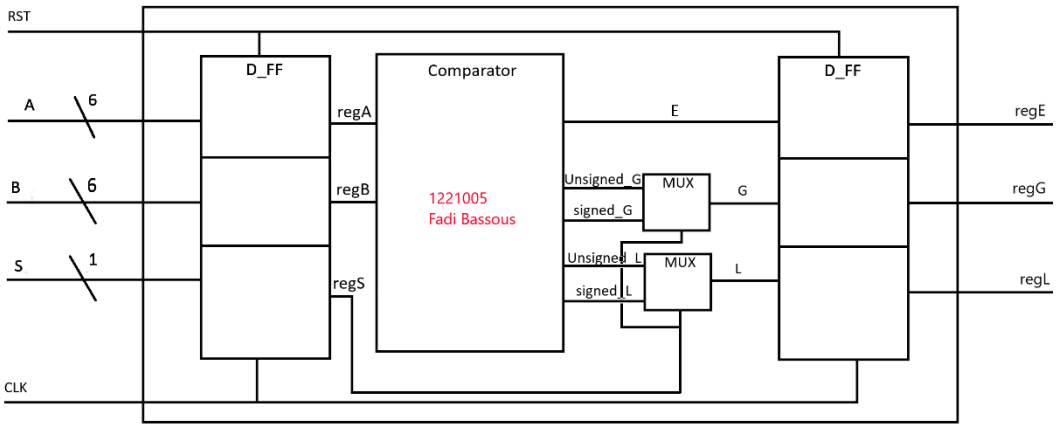

Figure 1: full diagram

Comparator

The comparator is designed to compare two 6-bit numbers and determine if they are equal, greater, or smaller. It supports both signed numbers, using 2's complement for negative values, and unsigned numbers, treating all values as positive. The comparator is built in a structural way using logic gates, with delays added to each gate to model realistic timing. A control signal selects the mode (signed or unsigned) through a 2-bit multiplexer. This design makes the comparator versatile and suitable for applications requiring accurate and reliable comparisons in digital systems.

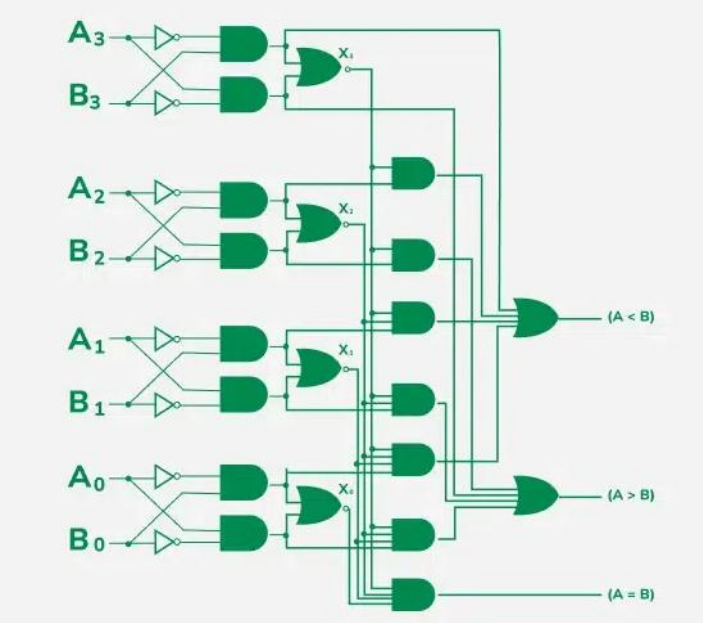

Figure 2: 4-bit comparator circuit

In the figure1 above, a 4-bit comparator circuit is shown. For my implementation, I adapted this structure to design a 6-bit comparator in my code.

Equal (E):
The "Equal" output is high when all corresponding bits of the two inputs, A and B, are identical. This is achieved using XNOR gates for each bit and an AND gate to combine all results.

Equation: $E=X[5]\cdot X[4]\cdot X[3]\cdot X[2]\cdot X[1]\cdot X[0]$

where $X[i]=A[i]\oplus B[i]$

Less Than (L):
The "Less Than" output is high when the first position where A and B differ indicates that A<B.

checking if both G (A > B) and E (A == B) are zero using an XOR gate. The XOR produces 1 when neither condition is true, indicating A < B. A 2-to-1 multiplexer (mux_2to1) then assigns 1 to unsigned_L when this condition is met and 0 otherwise, completing the less-than comparison.

Greater Than (G):
The "Greater Than" output is high when the first position where A and B differ indicates that A>B.

This is determined using NOT and AND gates similarly to Less Than logic.
Equation:
$G=(A[5]\cdot\sim B[5])+(A[4]\cdot\sim B[4]\cdot X[5])+(A[3]\cdot\sim B[3]\cdot X[5]\cdot X[4])+...+(A[0]\cdot\sim B[0]\cdot X[5]\cdot X[4]\cdot X[3]\cdot X[2]\cdot X[1])$

Signed and Unsigned Numbers:

Unsigned numbers are always positive and treated as regular binary values. Signed numbers can be positive or negative and are represented using 2's complement, where the most significant bit (MSB) indicates the sign (0 for positive, 1 for negative). This difference in representation changes how comparisons are performed, especially for negative numbers.

Handling Signed Numbers in the Code:

The code handles signed numbers by using the most significant bit (MSB) to determine whether A or B is positive or negative. XOR gates detect sign differences (V), and additional logic ensures correct comparisons. For the "Less Than" (L) output, the code checks if A is negative and B is positive (F) or if their signs differ (W), combining this with the unsigned result (unsigned L) using XOR. Similarly, for "Greater Than" (G), the signed result (signed G) adjusts the unsigned comparison (unsigned G) based on the sign logic. This ensures accurate signed number comparisons in the comparator.

Multiplexer

The multiplexer is used to select between signed and unsigned comparison modes. It is implemented in a structural way using basic logic gates, with delays added to each gate to reflect realistic timing behavior.

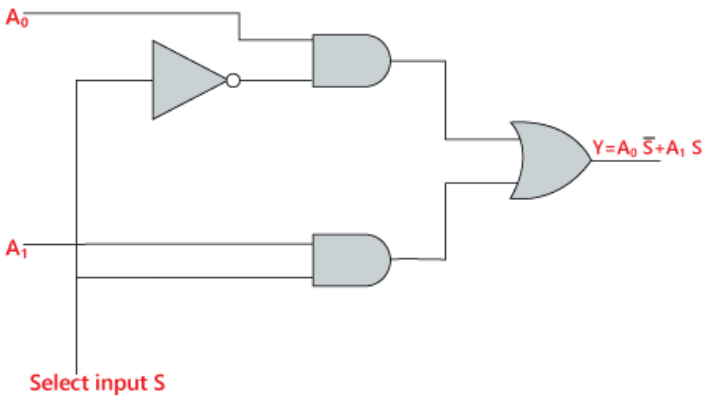

Figure 3: 2×1 Multiplexer circuit

input	output
S	A
0	unsigned
1	signed

Table 1: Multiplexer with Truth Table:

D Flip-Flop (registers)

D flip-flops are used as registers to store the inputs A and B and the outputs E, G, and L. These registers ensure the comparator works synchronously with the clock signal, making the circuit stable and reliable.

The registers are important for keeping the circuit organized and ensuring it operates correctly in a sequential system.

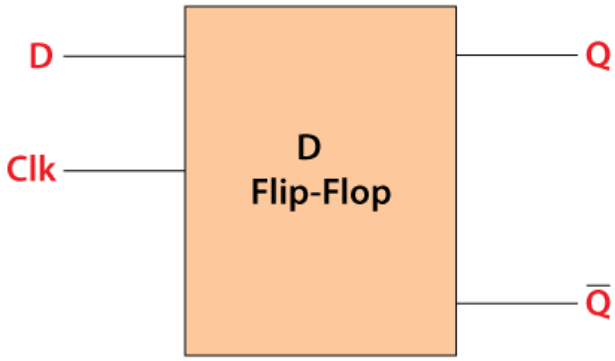

Figure 4: D Flip-Flop

Testbench

The testbench verifies the comparator's functionality by generating random 6-bit values for A and B and a control signal S to switch between signed and unsigned modes. It compares the outputs of the structural design with a behavioral model to check for errors, ensuring the comparator works correctly for all input combinations and detects any design issues.

Results:

No error

In figure 4 shows the result of the simulation of the comparator design. It generated a number of randomly chosen test cases for the 6-bit inputs A and B, along with the control signal S for switching between the signed (S=1) and unsigned (S=0) modes. In each case, all outputs—Equal, Less, and Greater—are correct; hence, this design has no bugs. The last message says simulation successfully completed with no problems detected.

```
# KERNEL: ASDB file was created in location C:\Users\Fadi Bassous\Desktop\BZU\y3s1\Advanced digital\
#  1:53 AM, Friday, December 27, 2024
#  Simulation has been initialized
run
# KERNEL: PASS: Time: 300 | A = 000010, B = 000010, S = 0 | Equal = 1, Less = 0, Greater = 0
# KERNEL: PASS: Time: 480 | A = 010010, B = 010010, S = 1 | Equal = 1, Less = 0, Greater = 0
# KERNEL: PASS: Time: 660 | A = 100100, B = 000001, S = 1 | Equal = 0, Less = 1, Greater = 0
# KERNEL: PASS: Time: 840 | A = 100011, B = 001101, S = 1 | Equal = 0, Less = 1, Greater = 0
# KERNEL: PASS: Time: 1020 | A = 100101, B = 010010, S = 1 | Equal = 0, Less = 1, Greater = 0
# KERNEL: PASS: Time: 1200 | A = 001101, B = 110110, S = 1 | Equal = 0, Less = 0, Greater = 1
# KERNEL: PASS: Time: 1380 | A = 101101, B = 001100, S = 1 | Equal = 0, Less = 1, Greater = 0
# KERNEL: PASS: Time: 1560 | A = 000110, B = 000101, S = 0 | Equal = 0, Less = 0, Greater = 1
# KERNEL: PASS: Time: 1740 | A = 100101, B = 110111, S = 0 | Equal = 0, Less = 1, Greater = 0
# KERNEL: PASS: Time: 1920 | A = 001111, B = 110010, S = 0 | Equal = 0, Less = 1, Greater = 0
# KERNEL: PASS: Time: 2100 | A = 101000, B = 000101, S = 0 | Equal = 0, Less = 0, Greater = 1
# KERNEL: PASS: Time: 2280 | A = 111101, B = 101101, S = 1 | Equal = 0, Less = 0, Greater = 1
# KERNEL: SIMULATION COMPLETE: No errors detected.
```

Figure 5: Output with no error

In figure 5 shows part of the code in Verilog. This chunk here is correct. In the next case i will change the Y_G[0] (in red circle) to Y_G[1] to make an error.

```
// Unsigned G = A > B
and #7 (Y_G[5], regA[5], notB[5]);
and #7 (Y_G[4], regA[4], notB[4], X[5]);
and #7 (Y_G[3], regA[3], notB[3], X[5], X[4]);
and #7 (Y_G[2], regA[2], notB[2], X[5], X[4], X[3]);
and #7 (Y_G[1], regA[1], notB[1], X[5], X[4], X[3], X[2]);
and #7 (Y_G[0], regA[0], notB[0], X[5], X[4], X[3], X[2], X[1]);
or  #7 (unsigned_G, Y_G[5], Y_G[4], Y_G[3], Y_G[2], Y_G[1], Y_G[0]);

// Signed L = A < B for signed numbers
    xor #10 (V,regA[5],regB[5]);
```

Figure 6: part of the code (no error)

With error

in figure 6 below is the result of a simulation, whereby an error has been introduced in the design. Due to this, certain test cases show wrong outputs, as indicated by "ERROR" messages. A testbench would correctly highlight these errors by comparing structural design outputs with expected outputs from a behavioral model, thus proving its effectiveness.

```
#  1:55 AM, Friday, December 27, 2024
#  Simulation has been initialized
run
# KERNEL: ERROR: Time: 300 | A = 000010, B = 000010, S = 0 | uut: E=1, L=0, G=x | uut2: E=1, L=0, G=0
# KERNEL: ERROR: Time: 480 | A = 010010, B = 010010, S = 1 | uut: E=1, L=0, G=x | uut2: E=1, L=0, G=0
# KERNEL: PASS: Time: 660 | A = 100100, B = 000001, S = 1 | Equal = 0, Less = 1, Greater = 0
# KERNEL: PASS: Time: 840 | A = 100011, B = 001101, S = 1 | Equal = 0, Less = 1, Greater = 0
# KERNEL: PASS: Time: 1020 | A = 100101, B = 010010, S = 1 | Equal = 0, Less = 1, Greater = 0
# KERNEL: ERROR: Time: 1200 | A = 001101, B = 110110, S = 1 | uut: E=0, L=0, G=x | uut2: E=0, L=0, G=1
# KERNEL: PASS: Time: 1380 | A = 101101, B = 001100, S = 1 | Equal = 0, Less = 1, Greater = 0
# KERNEL: ERROR: Time: 1560 | A = 000110, B = 000101, S = 0 | uut: E=0, L=0, G=x | uut2: E=0, L=0, G=1
# KERNEL: ERROR: Time: 1740 | A = 100101, B = 110111, S = 0 | uut: E=0, L=1, G=x | uut2: E=0, L=1, G=0
# KERNEL: ERROR: Time: 1920 | A = 001111, B = 110010, S = 0 | uut: E=0, L=1, G=x | uut2: E=0, L=1, G=0
# KERNEL: PASS: Time: 2100 | A = 101000, B = 000101, S = 0 | Equal = 0, Less = 0, Greater = 1
# KERNEL: PASS: Time: 2280 | A = 111101, B = 101101, S = 1 | Equal = 0, Less = 0, Greater = 1
# KERNEL: SIMULATION COMPLETE: 6 errors detected.
```

Figure 7: Output with error

In figure 7 represents part of the Verilog code. This chunk here is not correct. I changed Y_G[0] to Y_G[1] (in red circle) to make an error (shown in figure 6) to test the module

```
// Unsigned G = A > B
and #7 (Y_G[5], regA[5], notB[5]);
and #7 (Y_G[4], regA[4], notB[4], X[5]);
and #7 (Y_G[3], regA[3], notB[3], X[5], X[4]);
and #7 (Y_G[2], regA[2], notB[2], X[5], X[4], X[3]);
and #7 (Y_G[1], regA[1], notB[1], X[5], X[4], X[3], X[2]);
and #7 (Y_G[1], regA[0], notB[0], X[5], X[4], X[3], X[2], X[1]);
or  #7 (unsigned_G, Y_G[5], Y_G[4], Y_G[3], Y_G[2], Y_G[1], Y_G[0]);

// Signed L = A < B for signed numbers
    xor #10 (V,regA[5],regB[5]);
```

Figure 8: part of the code (with error)

conclusion

The project successfully designed and validated a 6-bit comparator capable of processing both signed and unsigned numbers. By implementing the design structurally with basic logic gates, the system achieved modularity, clarity, and synchronization via registers. The testbench played a crucial role in verifying functionality by generating random test cases and comparing the structural design outputs with a behavioral model. Deliberate introduction of errors confirmed the reliability of the testbench in identifying discrepancies. Simulation results verified the comparator's accuracy for all valid input combinations. This project establishes a strong foundation for future improvements, such as reducing delays, increasing bit-width, and integrating advanced error-detection techniques.

Future Work

1. Optimize gate delays to reduce overall latency.
2. Extend the design to support higher bit-widths (e.g., 8-bit or 16-bit).
3. Integrate advanced error detection techniques for enhanced reliability.
4. Investigate advanced synchronization methods using clock gating
5. Investigate the use of advanced register configurations to minimize propagation delays and enhance system performance