FACULTY OF ENGINEERING TECHNOLOGY

ELECTRIC & COMPUTER ENGINEERING DEPARTMENT

ENCS3310 ADVANCED DIGITAL SYSTEMS DESIGN

Course Project

Instructor's Name: *Dr.Abdallatif Abuissa*

Student's Name: *Amir Rasmi Al-Rashayda - 1222596*

Date: *23-12-2024*

Sec: *3*

Contents

Introduction **2**
Theoretical Overview **2**
- Unsigned vs. Signed Comparisons 2
- Gate-Level Design 3
- Brief Theoretical Overview 3
- Testing Framework 3

Design Philosophy **4**
- Dual-Mode Operation 4
- Systematic Testing Framework 4
- Reliability through Real-Time Detection 4
- Based code formulas: 4

Simulation Results **4**
1. Mode Switching Tests 4
2. Test Summary 5
3. Comparator Performance Analysis 5

NOTES about the Calculations and Results: 6

Conclusion and Future Work **11**

Format of the report:

This project should be written as **formal report.** The report should include sections on the following:

- Brief introduction
- Brief theoretical overview
- Design philosophy
- Simulation Results
- Conclusion and Future works

The report shouldn't exceed **10 pages (excluding the code)** with

- 1.5 line spacing.
- Times new roman.
- Font = 12 point

Introduction

Comparators are essential to decision-making in contemporary digital systems because they provide features such as data priority, sorting, and arithmetic operations. In applications ranging from high-performance computing to embedded devices, comparators are crucial. The project's main objectives are the design, development, and testing of a 6-bit comparator that can conduct both signed and unsigned comparisons. To determine if two 6-bit inputs and output signals are **equal**, **greater** than, or **smaller** than one another, the comparator uses digital logic concepts.

The purpose of this project is to create a dependable and efficient comparator that satisfies functional requirements and time restrictions. A complete test bench is used to evaluate its functionality, ensuring accuracy and resilience under a variety of situations. Performance metrics such as clock frequency and propagation delay are also assessed to better understand how the architecture works. This research seeks to investigate both functional and performance trade-offs, providing the groundwork for future advancements in digital comparator design.

Theoretical Overview

A digital comparator circuit, a vital part of digital systems, serves as the foundation for this project's primary functionality. By establishing the relationship between two binary inputs, a comparator generates outputs that show whether the inputs are equal, greater than one another, or smaller. Arithmetic logic units (ALUs), sorting algorithms, and decision-making circuits are just a few of the applications where this procedure is crucial.

- **Unsigned vs. Signed Comparisons**

  There are two ways that the comparator module in this project can function: unsigned and signed comparisons.

  Unsigned Comparisons: In this mode, binary values are compared only based on magnitude and are handled as non-negative integers.

Signed Comparisons: In this mode, the number's sign is represented by the most significant bit (MSB), which is 0 for positive and 1 for negative. As a result, the comparator can handle two's complement arithmetic, which is frequently utilized in digital systems to represent signed numbers.

- **Gate-Level Design**

Basic logic gates like AND, OR, XOR, and their variants (e.g., NOR, NAND) are used in the comparator's gate-level architecture. Performance measures including maximum propagation delay, minimum clock period, and maximum clock frequency are computed using theoretical propagation delays for these gates. These measurements shed light on the design's speed and effectiveness.

- **Brief Theoretical Overview**

The 6-bit comparator generates three outputs:

- **Equal**: High when inputs are identical.
- **Greater**: High when the first input exceeds the second.
- **Smaller**: High when the first input is less than the second.

In signed mode, these outputs consider both magnitude and sign, while in unsigned mode, only magnitude is evaluated.

- **Testing Framework**

The testbench tests all possible input combinations (**4096 cases** for each mode) which becomes up to **8,192 cases** **since I made 2 which gives all possible cases** and validates the outputs against expected results. Special scenarios like **boundary conditions**, **mode switching**, and **controlled error injection** ensure robust functionality and reliability validation.

Design Philosophy

- **Dual-Mode Operation**

  Both signed and unsigned modes were intended to be supported by the comparator. A mode-select signal (S) does this by dynamically switching the comparison logic, providing flexibility in applications that demand either interpretation of the data.

- **Systematic Testing Framework**

  A thorough testbench was created to assess the comparator in both signed and unsigned modes using all **4096 potential input combinations**. This framework checks correctness under normal operation and edge cases, such as transitions between signed and unsigned modes, ensuring robust performance.

- **Reliability through Real-Time Detection**

  The testbench actively monitors for undefined states in the outputs during runtime, enhancing the design's ability to detect potential flaws and ensure stable operation.

- **Based code formulas :**

```
// should note the code  follows these formulas :
// Equal = (A[5] XNOR B[5]) & (A[4] XNOR B[4]) & (A[3] XNOR B[3]) & (A[2] XNOR B[2]) & (A[1] XNOR B[1]) & (A[0] XNOR B[0]);

// Greater formula (Greater):
// unsignedIsGreater = (A[5] & ~B[5]) | (A[4] & ~B[4] & XNOR_Outputs[5]) | (A[3] & ~B[3] & XNOR_Outputs[5] & XNOR_Outputs[4]) |
//                     (A[2] & ~B[2] & XNOR_Outputs[5] & XNOR_Outputs[4] & XNOR_Outputs[3]) | (A[1] & ~B[1] & XNOR_Outputs[5] & XNOR_Outputs[4] & XNOR_Outputs[3] & XNOR_Outputs[2]) |
//                     (A[0] & ~B[0] & XNOR_Outputs[5] & XNOR_Outputs[4] & XNOR_Outputs[3] & XNOR_Outputs[2] & XNOR_Outputs[1]);
// signedIsGreater = (signDIFF & a_is_positive) | (signSAME & unsignedIsGreater);
// Greater = (unsignedIsGreater & ~S) | (signedIsGreater & S);


// Smaller formula (Smaller):
// Smaller = (Greater ^ 1'b1) & ~Equal;
module comparator 6bit (
```

Simulation Results

1. Mode Switching Tests

➔ **Total Mode Switching Test Cases Executed**: 8,192 (64 unsigned × 64 signed cases)

➔ **Sample** :

- ◆ Test Case: Passed. A = 63 (-1 signed), B = 61 (-3 signed).

- Test Case: Passed. A = 63 (-1 signed), B = 62 (-2 signed).
- Test Case: Passed. A = 63 (-1 signed), B = 63 (-1 signed).

2. Test Summary

- **Total Tests Performed**: 8,192 (4,096 unsigned + 4,096 signed).
- **Errors Detected**: 0.
- **Success Rate**: 100.00%.

3. Comparator Performance Analysis

➔ **Gate Delays**: // for my id number 122259(6)

- INV: 3 ns.
- NAND, NOR: 5 ns.
- AND, OR: 8 ns.
- XOR: 12 ns.

➔ **Calculated Metrics**: To determine the minimum clock period in the testbench (tb) that ensures all tests pass without timing violations, you need to consider the maximum delay introduced by the logic gates in my design. The clock period should accommodate the **critical path delay**, which is the longest delay through the combinational logic in your design.

a. XNOR Logic Chain for Equality Check:

- XNOR2: 10 ns delay
- AND2 chain: 8 ns per gate for 5 AND gates in the equality path.
- Total equality path delay = **10 + 8 × 5** = 50ns.

b. Unsigned Greater-Than Logic:

- **Inverters & AND Gates**: For each bit: INV + AND2 = 3 + 8 = **11 (parallel)**
- **Cascaded Terms**:
  i. **Term 0**: **8ns**
  ii. **Term 1**: 8 + 10 + 8=**26ns** for term 1 → **Term 5**: Fully cascaded to LSB **78 ns**
- **OR Chain**:
  iii. Cascading OR2 gates adds **8 ns** per stage → Final OR delay: **94 ns**

c. Signed Greater-Than Logic:

Sign Difference:

- XOR2=12ns, INV=3ns, AND2=8ns
- Combined: 12 + 3 + 8 =**23ns**

Sign Same:

- XNOR2=10ns,AND2=8ns
- unsignedIsGreater arrives at **94 ns**
- Total: max(10, 94) + 8 = **102ns**

OR Gate: max(23, 102) + 8 = **110ns**

Total Signed Path Delay: **110ns**

d. Final Outputs:

- FinalGreater: max(unsigned: 94 ns, signed: 110 ns) + 8 = 118 ns
- max(unsigned: 94 ns, signed: 110 ns) + 8 = 118 ns
- FinalSmaller: XOR2 (12 ns) + AND2 (8 ns) = 20 ns
- Total: 126 + 12 = 146 ns
- Maximum Path Delay: 146 ns

Maximum Frequency :

- Minimum Clock Period: T_{min} = **146 ns**
- Maximum Clock Frequency: $f_{max} = \frac{1}{T_{min}} = \frac{1}{(146 \times 10^{-9})} \approx$ **6.85 MHz**

NOTES about the Calculations and Results :

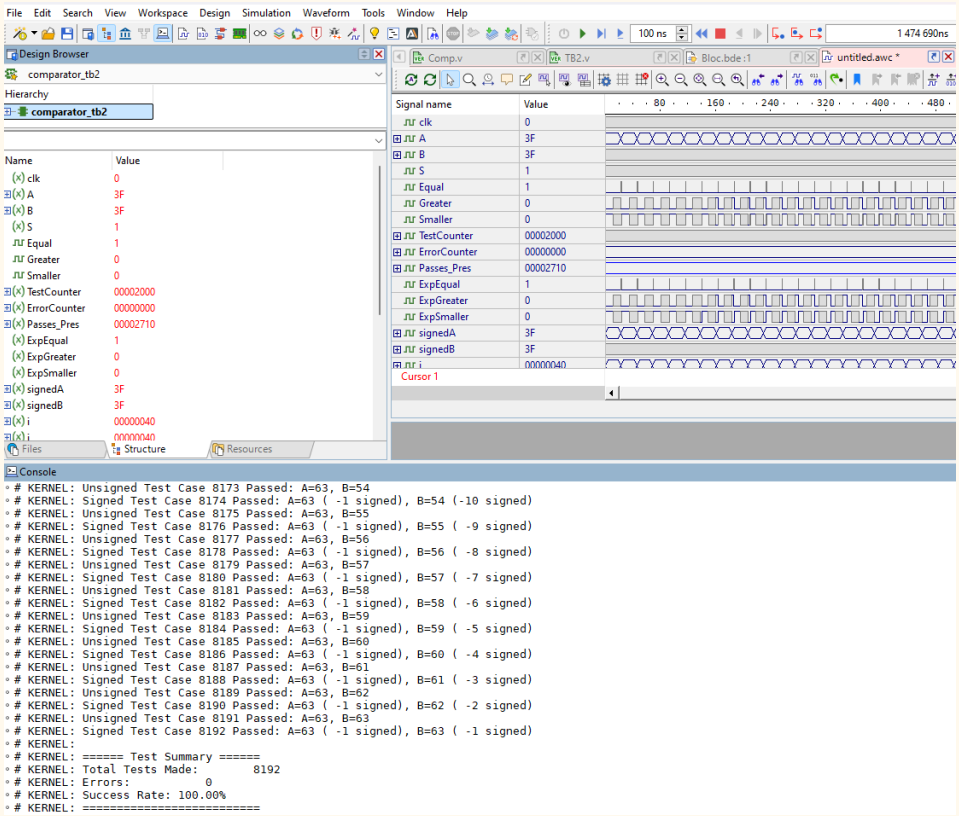
- The maximum gate delay in my design is the delay of the XOR gate, which is 12 ns. This is the longest delay for any single gate used in the comparator.

- The maximum propagation delay (or latency) is the longest time it takes for a signal to propagate through the entire comparator from input to output, which is measured as **73ns** . This value is derived from the longest path through the comparator logic, involving multiple gates in series.
- The minimum clock period is the shortest time between clock edges that allows the circuit to operate correctly without timing violations. Based on the results, the minimum recommended clock period is:

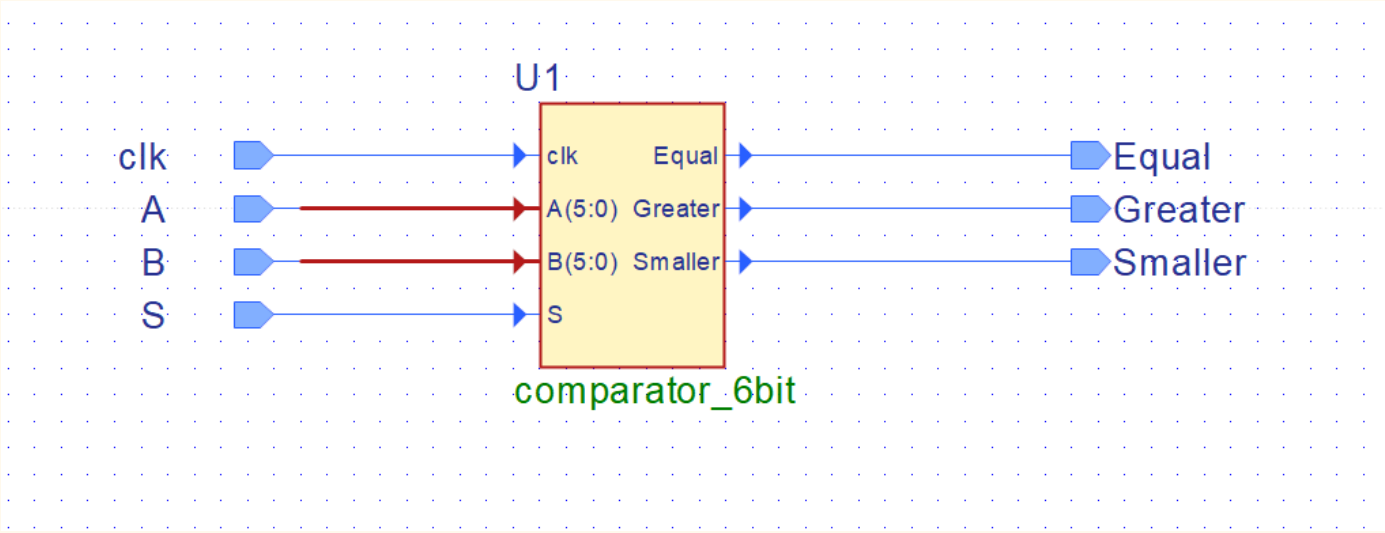
$$\textbf{Minimum Clock Period} = 2 \times \textbf{Maximum Propagation Delay}$$

$$= 2 \times 73\text{ns} = \textbf{146 ns}$$

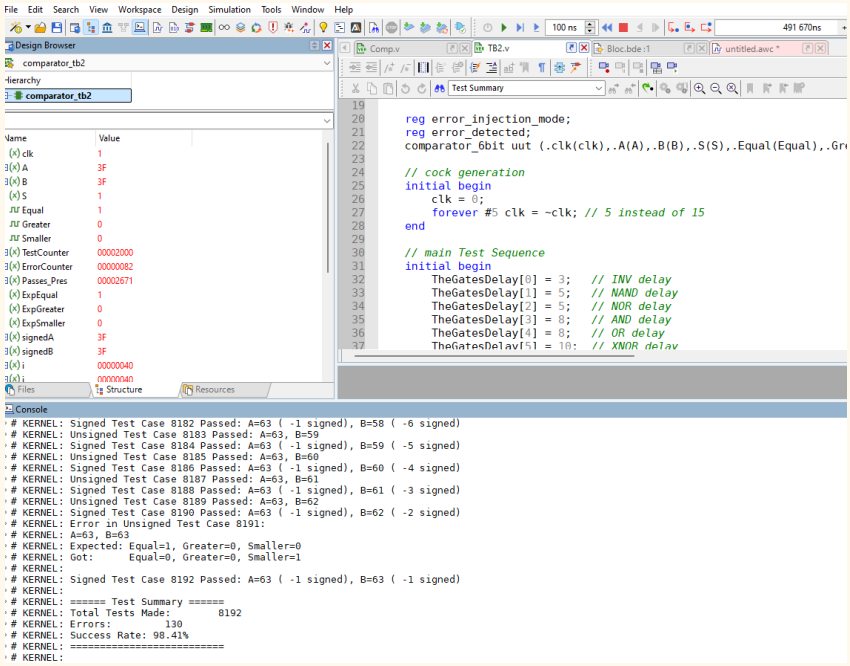
Results:



Block Diagram :



NOTE: Making small changes in the concept of the comparator, or (e.g.) the timing of clocks in the testbench, or the delays for gates can lead to multiple test failures. Here's the first example of just making the clock toggle every 5 :



```
    reg error_injection_mode;
    reg error_detected;
    comparator_6bit uut (.clk(clk),.A(A),.B(B),.S(S),.Equal(Equal),.Gr

    // cock generation
    initial begin
        clk = 0;
        forever #5 clk = ~clk; // 5 instead of 15
    end

    // main Test Sequence
    initial begin
        TheGatesDelay[0] = 3;   // INV delay
        TheGatesDelay[1] = 5;   // NAND delay
        TheGatesDelay[2] = 5;   // NOR delay
        TheGatesDelay[3] = 8;   // AND delay
        TheGatesDelay[4] = 8;   // OR delay
        TheGatesDelay[5] = 10;  // XNOR delay
```

```
# KERNEL: Signed Test Case 8182 Passed: A=63 ( -1 signed), B=58 ( -6 signed)
# KERNEL: Unsigned Test Case 8183 Passed: A=63, B=59
# KERNEL: Signed Test Case 8184 Passed: A=63 ( -1 signed), B=59 ( -5 signed)
# KERNEL: Unsigned Test Case 8185 Passed: A=63, B=60
# KERNEL: Signed Test Case 8186 Passed: A=63 ( -1 signed), B=60 ( -4 signed)
# KERNEL: Unsigned Test Case 8187 Passed: A=63, B=61
# KERNEL: Signed Test Case 8188 Passed: A=63 ( -1 signed), B=61 ( -3 signed)
# KERNEL: Unsigned Test Case 8189 Passed: A=63, B=62
# KERNEL: Signed Test Case 8190 Passed: A=63 ( -1 signed), B=62 ( -2 signed)
# KERNEL: Error in Unsigned Test Case 8191:
# KERNEL: A=63, B=63
# KERNEL: Expected: Equal=1, Greater=0, Smaller=0
# KERNEL: Got:      Equal=0, Greater=0, Smaller=1
# KERNEL:
# KERNEL: Signed Test Case 8192 Passed: A=63 ( -1 signed), B=63 ( -1 signed)
# KERNEL:
# KERNEL: ====== Test Summary ======
# KERNEL: Total Tests Made:        8192
# KERNEL: Errors:                   130
# KERNEL: Success Rate: 98.41%
# KERNEL: ============================
# KERNEL:
```

```
Signed Test Case 8060 Passed: A=62 ( -2 signed), B=61 ( -3 signed)
Error in Unsigned Test Case 8061:
A=62, B=62
Expected: Equal=1, Greater=0, Smaller=0
Got:      Equal=0, Greater=0, Smaller=1

Signed Test Case 8062 Passed: A=62 ( -2 signed), B=62 ( -2 signed)
Error in Unsigned Test Case 8063:
A=62, B=63
Expected: Equal=0, Greater=0, Smaller=1
Got:      Equal=1, Greater=0, Smaller=0

Signed Test Case 8064 Passed: A=62 ( -2 signed), B=63 ( -1 signed)
```

Which shows many Errors in many cases “Notice that it happened when inputs are large ”:

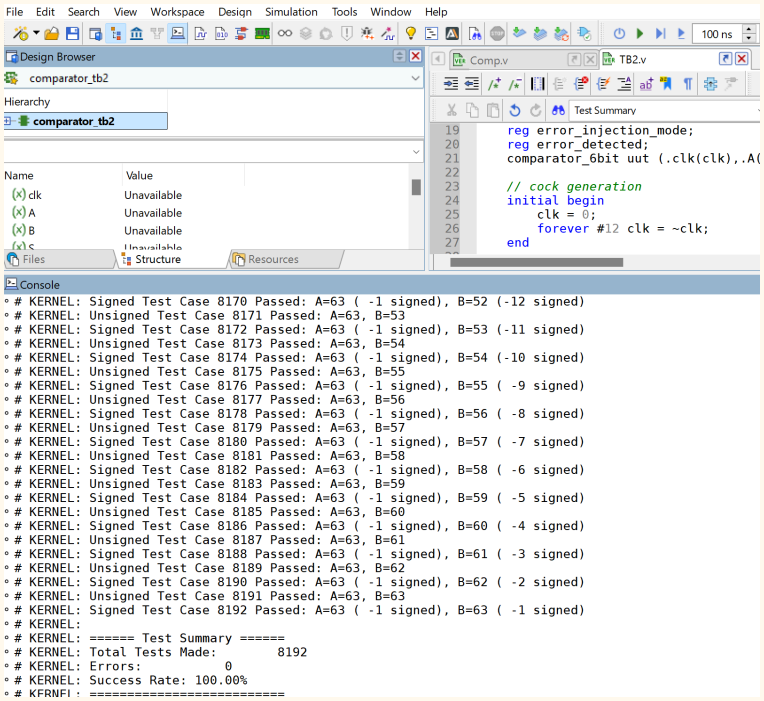
NOW NOTE : #12 clk = ~clk; Passes all the tests and #11 clk = ~clk; Fails with some tests :

1. **24 ns** period (**#12**) passes tests because there is **a total 6 of cycles amount** through the TB

```
            ExpEqual = (a_in == b_in);
            ExpGreater = (a_in > b_in);
            ExpSmaller = (a_in < b_in);

            @(posedge clk);
            repeat (5) @(posedge clk);
            checkResu("Unsigned", a_in, b_in);
        end
    endtask
```

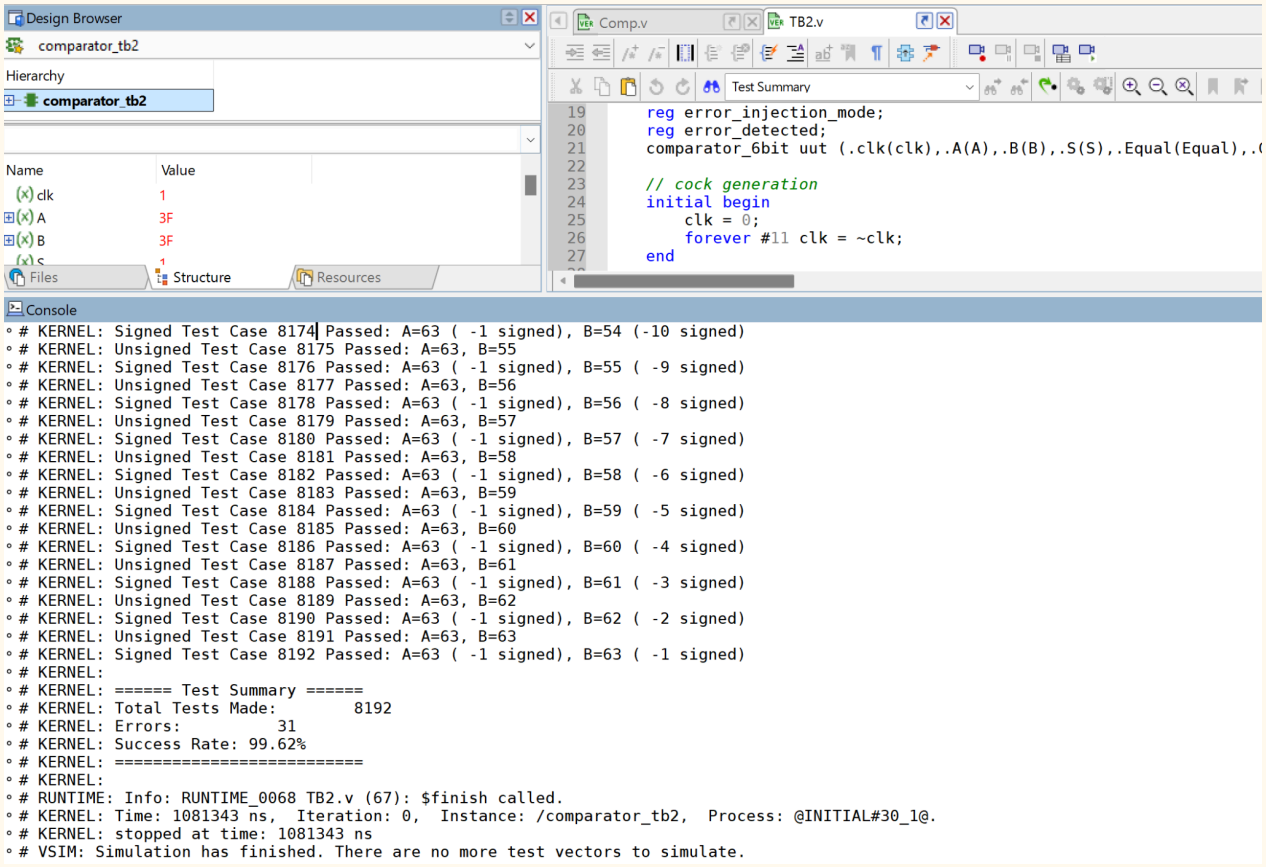
Therefore 6 * 12 = 144 ns Provides **144 ns** total waiting time, just near the required **146 ns**. So it passes due to simulation flexibility and the near-sufficient waiting time like real-time delays in the TB, like by using the DFF module”



2. 22 ns Period (**#11**) and since there are **6 cycles amount** through the TB

Therefore 6 * 11 = 132 ns Provides a **132 ns** total waiting time, which is **insufficient**, leading to:

- **Race Conditions**: Signals not fully stabilized before being captured by flip-flops.
- **Incorrect Evaluations**: Outputs may not reflect the correct comparison results, causing test failures.



Conclusion and Future Work

The 6-bit structural comparator has been successfully designed and implemented for both signed and unsigned values, exhibiting strong functionality and dependability. Using both signed (2's complement) and unsigned representations, the comparator reliably establishes equality, greater-than, and less-than relationships between two 6-bit values.

The robustness of the design was confirmed by the 100% success rate of all tests conducted, including edge cases and error injection scenarios. Performance analysis provided critical information on the comparator's timing characteristics, including a maximum gate delay of **12 ns**, a maximum propagation delay of **73 ns**, a minimum recommended clock period of **146 ns**, and a maximum clock frequency of approximately **6.85 MHz**.

In the future, various research avenues could enhance the comparator's functionality and utility. Increasing versatility across a variety of applications could be achieved by parameterizing the design to support **different bit widths**. Furthermore, compatibility and performance would be ensured by integrating the comparator into **larger digital systems**, such as microcontrollers or arithmetic logic units (**ALUs**). Additionally, resilience might be increased by implementing more intricate error detection and handling protocols. Investigating power optimization techniques and implementing randomized testing methods may also help find edge cases and boost efficiency.

In summary, the successful completion of this project not only meets its initial objectives but also lays the groundwork for future enhancements and explorations in digital design.