Faculty of Engineering and Technology

Electrical and Computer Engineering Department

ENCS3310 Project

Prepared by:

Basmala abu hakema **1220184**

Instructor: **Abdallatif Abuissa**

Section: **1**

Date: **24-12-2024**

Table of Contents:

Table of Figures:

Introduction:

The objective of this project is to design and implement a 6-bit comparator module capable of evaluating the relationship between two binary inputs (A and B) under both **unsigned** and **signed** comparison modes. The comparison produces three outputs: **A_greater_B**, **A_equal_B**, and **A_smaller_B**, which indicate whether A is greater than, equal to, or less than B, respectively. The module operates based on a Selection signal to toggle between signed and unsigned modes, making it versatile for various digital system applications. Additionally, flip-flops are utilized to store the input and output states, ensuring synchronous operation with a clock signal.

Theoretical overview:

Digital comparators:

Digital comparators are combinational circuits used to compare two binary numbers and determine their relationship. For two numbers A and B, the output is typically split into three categories:

- A>B: High if A is greater than B.
- A=B: High if A is equal to B.
- A<B: High if A is less than B.

Unsigned Comparison:

In unsigned comparison, the binary numbers are treated as non-negative integers. The most significant bit (MSB) has the highest weight in determining the magnitude of the number. For example if A=6’b100000 and B=6’b000001 then A>B.

Signed Comparison:

In signed comparison, numbers are represented using **two's complement** format. The MSB serves as the sign bit:

- 0: Positive number.
- 1: Negative number.

For two signed numbers A and B, their relationship depends on their signs:

1. A positive number is always greater than a negative number.
2. If both numbers have the same sign, their magnitudes are compared.

For example if A=6’b100000 and B=6’b000001 then A<B.

Design philosophy:

System Implementation

1. Input Handling and Storage

The system starts with two 6-bit inputs A and B representing the numbers to be compared. These inputs are stored in registers (register_6bit module) synchronized to the clock. The Selection signal determines which comparison mode (signed or unsigned) will be used.

2. Comparison Logic

2.1 Unsigned Comparison

For unsigned comparison, the system directly compares the corresponding bits of A and B using AND, NOR, and OR gates. This logic checks the relative magnitudes of the numbers by examining each bit, starting from the most significant bit (MSB). The result is determined by whether A is greater than, equal to, or smaller than B.

2.2 Signed Comparison

For signed comparison, the system first compares the signs of A and B (the MSBs). If the signs are different, the comparison is straightforward: the number with a 0 in the MSB is greater than the number with a 1. If the signs are the same, the system compares the magnitudes of the two numbers, using the five_bit_mag_comp module to compare the absolute values.

2.3 Multiplexer for Mode Selection

A 2-to-1 multiplexer with a selecation is used to select the appropriate comparison result from either the unsigned or signed comparator, based on the Selection signal. If the Selection is 0, the unsigned comparator's results are used. If the Selection is 1, the signed comparator's results are used.

3. Output Handling

The comparison results are stored in an output register (output_register), which updates on each clock cycle. The results are then assigned to the output signals A_greater_B, A_smaller_B, and A_equal_B, representing the final comparison results.

4. Simulation and Verification

Explanation of the Testbench

The **testbench** serves as a validation framework for the main_module, ensuring its correctness by comparing its outputs against a reference implementation (verification module). Here's a breakdown of its components:

Key Components

1. **Inputs and Outputs**:

    - Inputs (clock, A, B, Selection) are declared as reg, as they are driven by the testbench.

- Outputs (A_greater_B, A_smaller_B, A_equal_B) are declared as wire, as they are driven by the instantiated module.

2. **Module Instantiations**:
    - main_module: The Design Under Test (DUT) that implements the signed and unsigned comparison.
    - verification: A reference comparator module that validates the DUT's behavior.
3. **Clock Generation**:
    - The clock signal toggles every 100ns, ensuring synchronous operation for both DUT and reference modules.
4. **Testing Logic**:
    - **Nested Loops**:
        - Outer loop toggles Selection (0 for unsigned, 1 for signed).
        - Two inner loops iterate through all 64 combinations of 6-bit numbers A and B.
    - **Output Comparison**:
        - DUT outputs are compared with the reference module after every positive clock edge.
        - Discrepancies are logged with detailed information.
5. **Error and Test Tracking**:
    - total_tests: Tracks the total number of test cases executed.
    - error_count: Counts mismatched outputs between DUT and reference.
6. **Summary**:
    - A final summary is displayed, indicating the number of tests run and any errors found.
    - If no errors are encountered, "ALL TESTS PASSED!" is displayed.

Explanation of the Verification Module

The **verification module** serves as a reference implementation for comparison logic and its build behaviorally. It mimics the expected behavior of the DUT to validate its correctness.

Key Features

1. **Inputs and Outputs**:
    - Takes inputs A, B, and Selection, along with a clock signal (clk).
    - Outputs the comparison results (A_greater_B, A_equal_B, A_smaller_B).
2. **Internal Registers**:
    - Stores inputs (A and B) in internal registers (reg_A, reg_B) on the positive clock edge for synchronized processing.
3. **Comparison Logic**:
    - **Unsigned Comparison (Selection = 0)**:
        - Compares A and B directly using standard comparison operators.
    - **Signed Comparison (Selection = 1)**:
        - Uses $signed() casting to interpret A and B as signed numbers.
        - Compares values and sets the appropriate output signal.
4. **Clock-Driven Updates**:
    - Outputs are updated only on the positive clock edge, ensuring alignment with the testbench and DUT.

Block diagram:

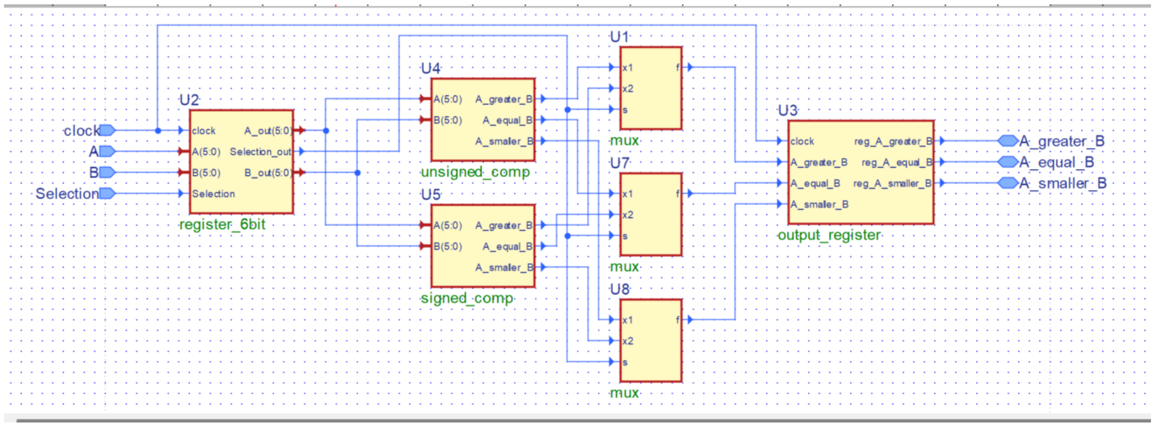
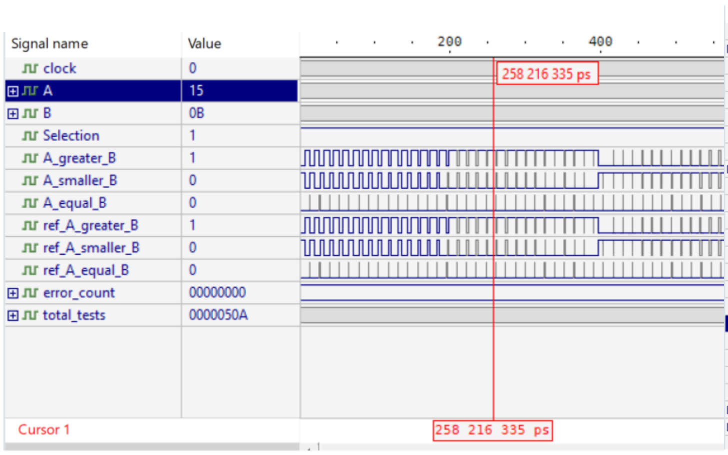
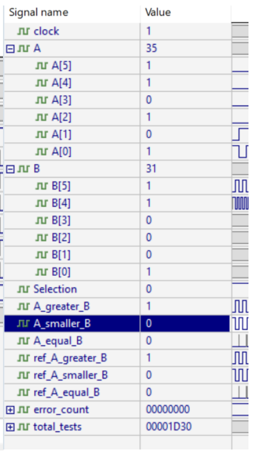


Figure 1: block diagram

Prove that the signed and unsigned compartor works:

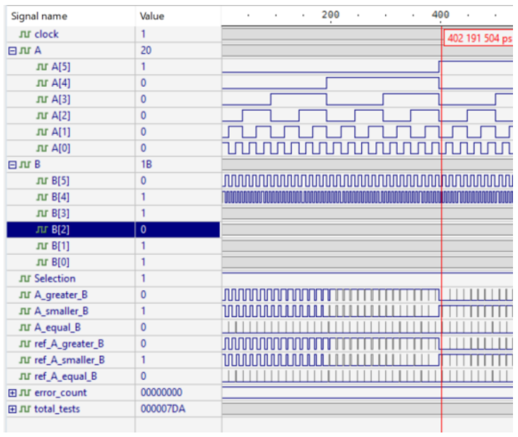


Signal name	Value
clock	0
A	15
B	0B
Selection	1
A_greater_B	1
A_smaller_B	0
A_equal_B	0
ref_A_greater_B	1
ref_A_smaller_B	0
ref_A_equal_B	0
error_count	00000000
total_tests	0000050A



Signal name	Value
clock	1
A	35
A[5]	1
A[4]	1
A[3]	0
A[2]	1
A[1]	0
A[0]	1
B	31
B[5]	1
B[4]	1
B[3]	0
B[2]	0
B[1]	0
B[0]	1
Selection	0
A_greater_B	1
A_smaller_B	0
A_equal_B	0
ref_A_greater_B	1
ref_A_smaller_B	0
ref_A_equal_B	0
error_count	00000000
total_tests	00001D30

Here selection is 1 so signed comparator and A>B   here selection is 0 so A>B

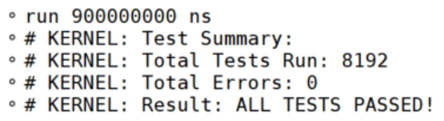


Signal name	Value
clock	1
A	20
A[5]	1
A[4]	0
A[3]	0
A[2]	0
A[1]	0
A[0]	0
B	1B
B[5]	0
B[4]	1
B[3]	1
B[2]	0
B[1]	1
B[0]	1
Selection	1
A_greater_B	0
A_smaller_B	1
A_equal_B	0
ref_A_greater_B	0
ref_A_smaller_B	1
ref_A_equal_B	0
error_count	00000000
total_tests	000007DA

Here selection is 1 so B > A

Simulation Results:

After running the testbench the result is:

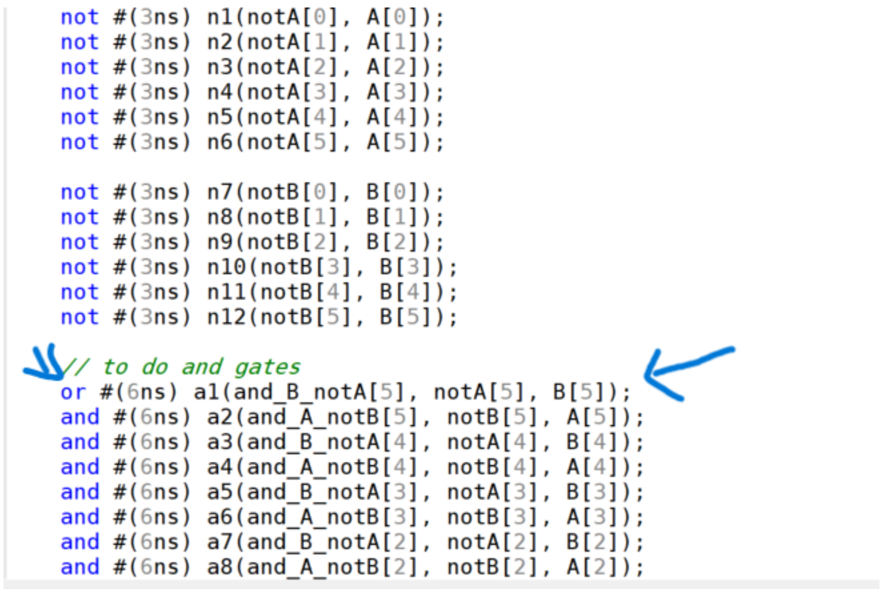


```
◦ run 900000000 ns
◦ # KERNEL: Test Summary:
◦ # KERNEL: Total Tests Run: 8192
◦ # KERNEL: Total Errors: 0
◦ # KERNEL: Result: ALL TESTS PASSED!
```

Figure 2: Passed results

Result: all tests passed.

If I change A and to an or the result gets failed:

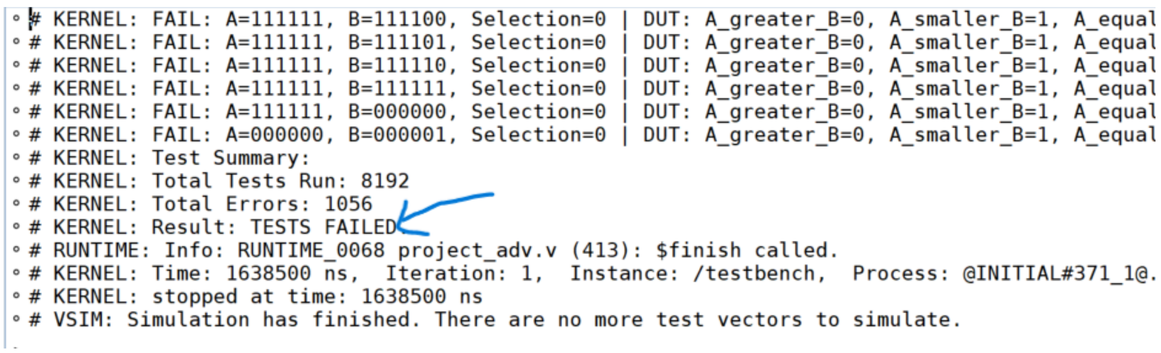


```
    not #(3ns) n1(notA[0], A[0]);
    not #(3ns) n2(notA[1], A[1]);
    not #(3ns) n3(notA[2], A[2]);
    not #(3ns) n4(notA[3], A[3]);
    not #(3ns) n5(notA[4], A[4]);
    not #(3ns) n6(notA[5], A[5]);

    not #(3ns) n7(notB[0], B[0]);
    not #(3ns) n8(notB[1], B[1]);
    not #(3ns) n9(notB[2], B[2]);
    not #(3ns) n10(notB[3], B[3]);
    not #(3ns) n11(notB[4], B[4]);
    not #(3ns) n12(notB[5], B[5]);

    // to do and gates
    or #(6ns) a1(and_B_notA[5], notA[5], B[5]);
    and #(6ns) a2(and_A_notB[5], notB[5], A[5]);
    and #(6ns) a3(and_B_notA[4], notA[4], B[4]);
    and #(6ns) a4(and_A_notB[4], notB[4], A[4]);
    and #(6ns) a5(and_B_notA[3], notA[3], B[3]);
    and #(6ns) a6(and_A_notB[3], notB[3], A[3]);
    and #(6ns) a7(and_B_notA[2], notA[2], B[2]);
    and #(6ns) a8(and_A_notB[2], notB[2], A[2]);
```

Figure 3: Failed results



```
◦ # KERNEL: FAIL: A=111111, B=111100, Selection=0 | DUT: A_greater_B=0, A_smaller_B=1, A_equal
◦ # KERNEL: FAIL: A=111111, B=111101, Selection=0 | DUT: A_greater_B=0, A_smaller_B=1, A_equal
◦ # KERNEL: FAIL: A=111111, B=111110, Selection=0 | DUT: A_greater_B=0, A_smaller_B=1, A_equal
◦ # KERNEL: FAIL: A=111111, B=111111, Selection=0 | DUT: A_greater_B=0, A_smaller_B=1, A_equal
◦ # KERNEL: FAIL: A=111111, B=000000, Selection=0 | DUT: A_greater_B=0, A_smaller_B=1, A_equal
◦ # KERNEL: FAIL: A=000000, B=000001, Selection=0 | DUT: A_greater_B=0, A_smaller_B=1, A_equal
◦ # KERNEL: Test Summary:
◦ # KERNEL: Total Tests Run: 8192
◦ # KERNEL: Total Errors: 1056
◦ # KERNEL: Result: TESTS FAILED
◦ # RUNTIME: Info: RUNTIME_0068 project_adv.v (413): $finish called.
◦ # KERNEL: Time: 1638500 ns,  Iteration: 1,  Instance: /testbench,  Process: @INITIAL#371_1@.
◦ # KERNEL: stopped at time: 1638500 ns
◦ # VSIM: Simulation has finished. There are no more test vectors to simulate.
```

So here the test failed because I changed and to an or.

Conclusion:

This Verilog code implements a comparator system that can compare two 6-bit numbers (A and B) both unsigned and signed, based on the Selection input. The main_module orchestrates the comparison, using two separate comparator modules: one for unsigned comparison (unsigned_comp) and one for signed comparison (signed_comp). The results are then selected based on the Selection signal, which determines whether the unsigned or signed comparison logic is used. Additionally, a 6-bit register stores the inputs, and another register stores the comparison results, with the final outputs (A_greater_B, A_smaller_B, A_equal_B) indicating the relationship between A and B. The testbench verifies the functionality by running all possible combinations of A and B for both unsigned and signed comparisons, comparing the module outputs to reference outputs, and logging any discrepancies.