K-VLSI_Cohort-02 – High Level Synthesis (KVLS602)

Assignment -01 Solution Sheet

Teacher – Prof. Subir Kumar Roy

Submission done by: -

Name: - Amit

Roll No: - KVLSI2501033

Solution to the First [A.] answer (4-bit magnitude comparator):

Objective:

To design a **4-bit magnitude comparator** using **structural Verilog**, which compares two 4-bit binary numbers `A` and `B` and determines whether:

- `A > B` → output: `A_gt_B`
- `A < B` → output: `A_lt_B`
- `A == B` → output: `A_eq_B`

A **magnitude comparator** compares two binary numbers **bit by bit**, starting from the **most significant bit (MSB)** to the **least significant bit (LSB)**.

Given:

- `A = A(3) A(2) A(1) A(0)`
- `B = B(3) B(2) B(1) B(0)`

An **intermediate variable** is defined for bit-wise equality check:

$x(i) = A(i)\cdot B(i) + A(i)'\cdot B(i)'$ → XNOR Function

Design Methodology:

This design is implemented using **structural Verilog**, specifically using **primitive gates** like `and`, `or`, and `xnor`.

Design Module Highlights:

- The **XNOR gates** compute `x[i]` to check bit-wise equality between `A` and `B`.
- Multiple **AND gates** generate comparison terms based on the above equations.
- The **OR gate** combines the terms to generate final outputs `A_gt_B` and `A_lt_B`.
- An **AND gate** chain computes `A_eq_B`.

The above approach is very much simple and upto mark which can be used to write the code for the design

Verilog design Code for the 4-bit magnitude Comparator

```
// Structural modelling style
module magnitude_comparator (
    input [3:0] A,  // 4-bit Input A
    input [3:0] B,  // 4-bit Input B
    output A_gt_B,  // Output: A Greater than B
    output A_lt_B,  // Output: A Less than B
    output A_eq_B ); // Output: A Equal to B

    // Intermediate XNOR results to check bit-wise equality
    wire [3:0] x;  // x(i) = A(i) XNOR B(i) = A(i).B(i) + A(i)'.B(i)'

    // Generate x[i] using XNOR gates (bit-wise equality)
    xnor (x[3], A[3], B[3]);
    xnor (x[2], A[2], B[2]);
    xnor (x[1], A[1], B[1]);
    xnor (x[0], A[0], B[0]);

    // Intermediate terms used in A_gt_B and A_lt_B logic
    wire term1, term2, term3, term4;
    wire term5, term6, term7, term8;

    // Logic for A > B
    // A_gt_B = A(3)·B(3)' + x(3)·A(2)·B(2)' + x(3)·x(2)·A(1)·B(1)' +
x(3)·x(2)·x(1)·A(0)·B(0)'
    and (term1, A[3], ~B[3]);                 // First condition: MSB comparison
    and (term2, x[3], A[2], ~B[2]);        // If MSBs are equal, compare next bit
    and (term3, x[3], x[2], A[1], ~B[1]);      // Next significant bit comparison
    and (term4, x[3], x[2], x[1], A[0], ~B[0]); // LSB comparison if all above equal
    or (A_gt_B, term1, term2, term3, term4);      // Combine all terms for A > B

    // Logic for A < B
    // A_lt_B = A(3)'·B(3) + x(3)·A(2)'·B(2) + x(3)·x(2)·A(1)'·B(1) +
x(3)·x(2)·x(1)·A(0)'·B(0)
    and (term5, ~A[3], B[3]);             // First condition: MSB comparison
    and (term6, x[3], ~A[2], B[2]);        // If MSBs are equal, compare next bit
    and (term7, x[3], x[2], ~A[1], B[1]);    // Next significant bit comparison
    and (term8, x[3], x[2], x[1], ~A[0], B[0]); // LSB comparison if all above equal
    or (A_lt_B, term5, term6, term7, term8);     // Combine all terms for A < B

    // Logic for A == B
    // A_eq_B = x(3)·x(2)·x(1)·x(0)
    and (A_eq_B, x[3], x[2], x[1], x[0]);   // All bits must be equal

endmodule
```

Testbench ideology :

The testbench `magnitude_comparator_tb`:

- Defines `A` and `B` as `reg` variables.
- Instantiating the module of 4-bit comparator
- Using the named mapping of variables and linking them to design code under DUT design unit under test
- Monitors the output values using `$monitor`.
- Dumps the waveform using `$dumpfile` and `$dumpvars`.

Applies multiple test vectors to verify all comparison cases: `A > B`, `A < B`, and `A == B`.

Test Vectors Used:

A (Binary)	B (Binary)	Description
0000	0000	A == B
1010	1001	A > B
0110	0111	A < B
1111	1111	A == B
0101	1000	A < B
1100	1011	A > B

Testbench Verilog code for the 4-bit magnitude comparator.

```
`timescale 1ns / 1ps


module magnitude_comparator_tb;
    reg [3:0] A, B;
    wire A_gt_B, A_lt_B, A_eq_B;

    // Instantiate the comparator module
    magnitude_comparator uut (.A(A),.B(B),.A_gt_B(A_gt_B),.A_lt_B(A_lt_B),.A_eq_B(A_eq_B));

    initial begin
        $dumpfile("magnitude_comparator.vcd");  // Generates waveform dump
        $dumpvars(0, magnitude_comparator_tb);

        // Test Cases
        A = 4'b0000; B = 4'b0000; #10;  // A == B
        A = 4'b1010; B = 4'b1001; #10;  // A > B
        A = 4'b0110; B = 4'b0111; #10;  // A < B
        A = 4'b1111; B = 4'b1111; #10;  // A == B
        A = 4'b0101; B = 4'b1000; #10;  // A < B
        A = 4'b1100; B = 4'b1011; #10;  // A > B
```

```
        $finish;
    end

    initial begin
        $monitor("Time=%0t A=%b B=%b | A_gt_B=%b A_lt_B=%b A_eq_B=%b",
                 $time, A, B, A_gt_B, A_lt_B, A_eq_B);
    end
endmodule
```

Simluation and Output:

Upon simulation:

- The comparator **accurately distinguishes** between greater, less, and equal conditions.
- Outputs are displayed in the terminal and available in waveform format (`.vcd`) for GTKWave viewing.

```
C:\Windows\System32\cmd.e
Microsoft Windows [Version 10.0.26100.3775]
(c) Microsoft Corporation. All rights reserved.

C:\iverilog\bin>vvp four_bc
VCD info: dumpfile magnitude_comparator.vcd opened for output.
Time=0 A=0000 B=0000 | A_gt_B=0 A_lt_B=0 A_eq_B=1
Time=10000 A=1010 B=1001 | A_gt_B=1 A_lt_B=0 A_eq_B=0
Time=20000 A=0110 B=0111 | A_gt_B=0 A_lt_B=1 A_eq_B=0
Time=30000 A=1111 B=1111 | A_gt_B=0 A_lt_B=0 A_eq_B=1
Time=40000 A=0101 B=1000 | A_gt_B=0 A_lt_B=1 A_eq_B=0
Time=50000 A=1100 B=1011 | A_gt_B=1 A_lt_B=0 A_eq_B=0
four_bit_comp_tb.v.txt:28: $finish called at 60000 (1ps)
```

Figure 1 Icarus Verilog Simulator output

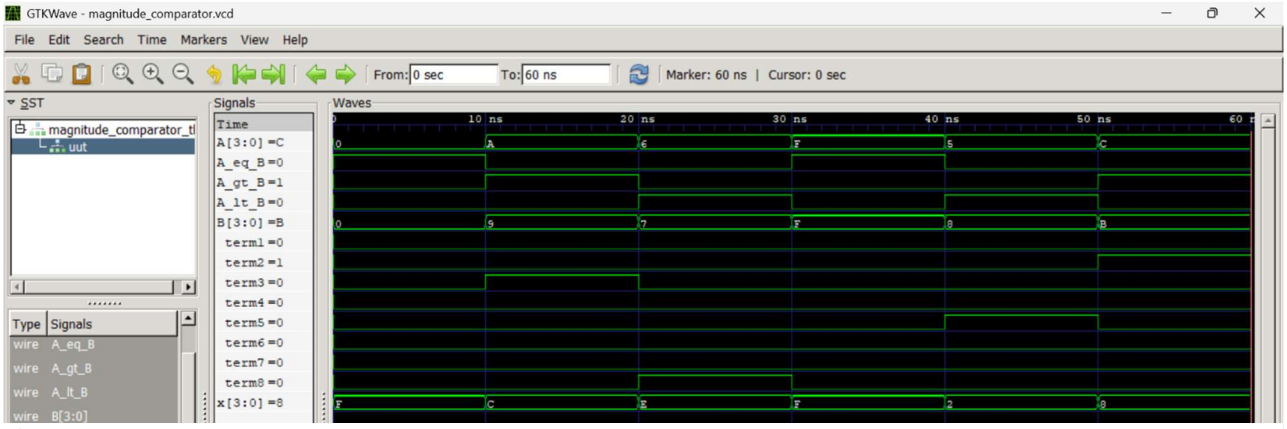

Figure 2 GTKWave waveform output

Conclusion:

This report successfully demonstrates the design and testing of a **4-bit magnitude comparator** using **structural Verilog modeling**. The comparator correctly evaluates the relative magnitude of two 4-bit numbers using **XNOR, AND, and OR** logic gates. The testbench verifies all possible output conditions, confirming the correctness of the logic implementation.

Links for the Assignment Solutions:-

GitHub Link:-
https://github.com/amitvsuryavanshi04/amit_kvlsi_iiitb/tree/1e84743552463ab2403cb7ecab78d198feaa6f6d/KVLS602-High%20Level%20Synthesis

EDA Playground Link: - https://edaplayground.com/x/Sj7X