LAB REPORT 4

2 bit COMPARATOR AND 4 BIT COMPARATOR
GROUP 13

GROUP MEMBERS: HASSAN AHMED

NEHA CHAUDHARY

HAMZA SHABBIR

SUBMITTED TO: Sir Asim

4.1 Objectives

The objectives of this lab are:

- We will design a 2-bit comparator schematic with the help of gate primitives
- We will then implement the 4- bit comparator using
    1. Instances using input and output arrays
- We will write a Verilog code for testing 2-bit and 4-bit comparators for 4 different inputs.

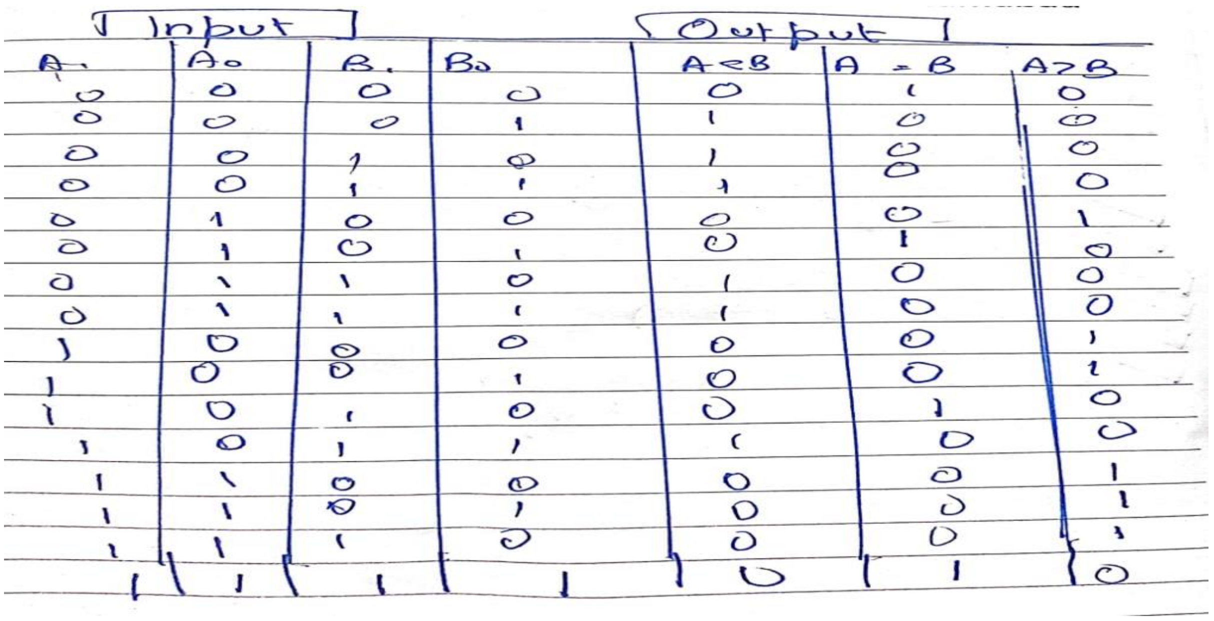
4.2 Background

A Comparator is a combinational circuit that returns the output as of A>B, A<B, and A=B.

2-Bit Magnitude Comparator

A comparator used to compare two binary numbers each of two bits is called a 2-bit Magnitude comparator. It consists of four inputs and three outputs to generate less than, equal to and greater than between two binary numbers.

TRUTH TABLE



Input				Output		
A_1	A_0	B_1	B_0	A<B	A = B	A>B
0	0	0	0	0	1	0
0	0	0	1	1	0	0
0	0	1	0	1	0	0
0	0	1	1	1	0	0
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	1	0	0
0	1	1	1	1	0	0
1	0	0	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	0	1	0
1	0	1	1	1	0	0
1	1	0	0	0	0	1
1	1	0	1	0	0	1
1	1	1	0	0	0	1
1	1	1	1	0	1	0

Using K-maps

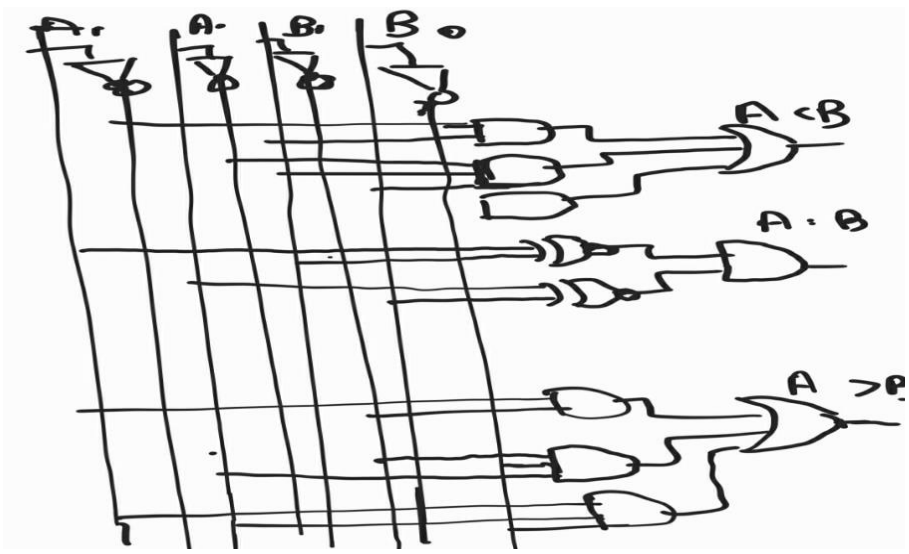
we get the following equations

$A>B = A1B1' + B0'(A0B1' + A0A1)$

$A<B = B1A1' + B0B1A0' + A1'A0'B0$

$A=B = (A0 \text{ Ex-Nor } B0)(A1 \text{ Ex-Nor } B1)$

DIAGRAM



4-Bit Magnitude Comparator

A comparator used to compare two binary numbers each of four bits is called a 4-bit magnitude comparator. It consists of eight inputs each for two four-bit numbers and three outputs to generate less than, equal to and greater than between two binary numbers.

The output logic statements of this converter are

If A3 = 1 and B3 = 0, then A is greater than B (A>B). Or

If A3 and B3 are equal, and if A2 = 1 and B2 = 0, then A > B. Or

If A3 and B3 are equal & A2 and B2 are equal, and if A1 = 1, and B1 = 0, then A>B. Or

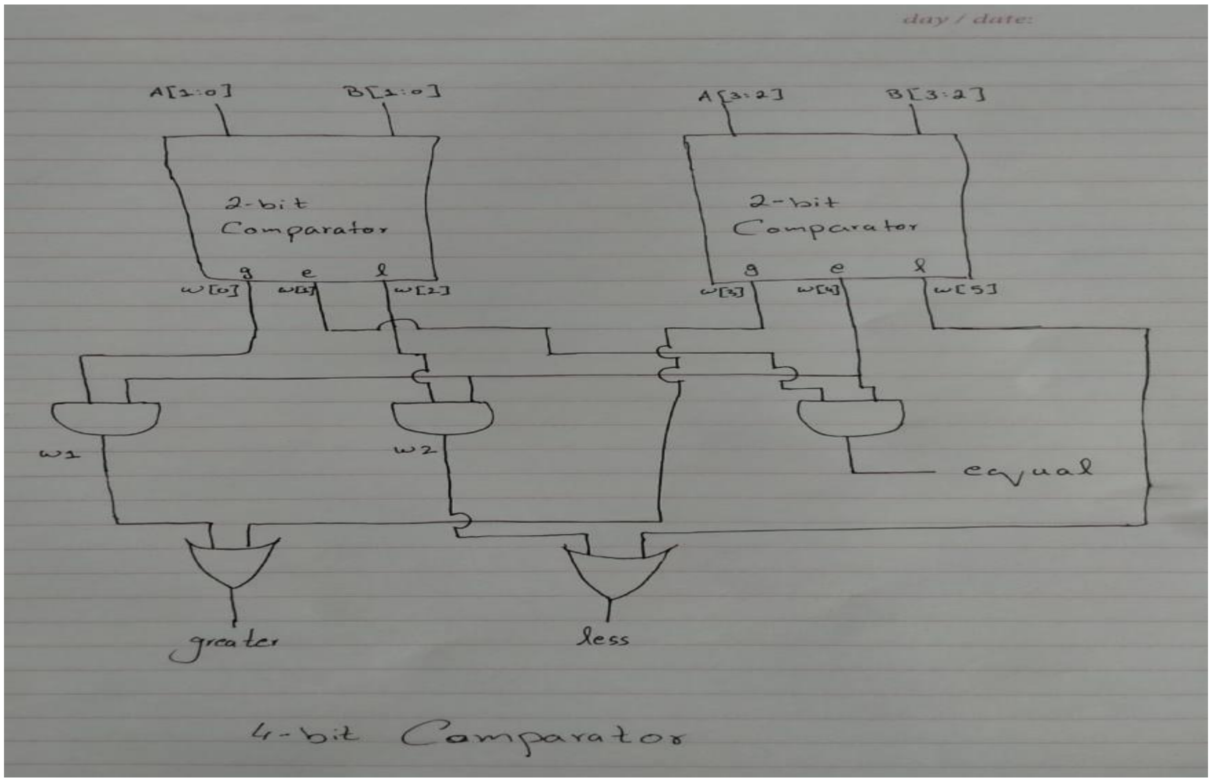
If A3 and B3 are equal, A2 and B2 are equal and A1 and B1 are equal, and if A0 = 1 and B0 = 0, then A > B.

From the above statements, the output A > B

The equal output is produced when all the individual bits of one number exactly coincide with corresponding bits of another number. Then the logical expression for A=B output can be written as

E = (A3 Ex-NOR B3) (A2 Ex-NOR B2) (A1 Ex-NOR B1) (A0 Ex-NOR B0)

DIAGRAM:



4.3 Equipment

- Computer / laptop
- ModelSim / ISE IDE software

4.5 Procedure

1. We implemented the 2-bit comparator schematic using gate primitives.

2. Then we verified the code using testbench for the 2-bit comparator

3. After that, we implemented a 4-bit comparator using 2 instances of the test benches20-time2-bit comparator.

4. 4-bit comparator was also verified using testbench.

4.6 Codes and Simulations

2-Bit Comparator

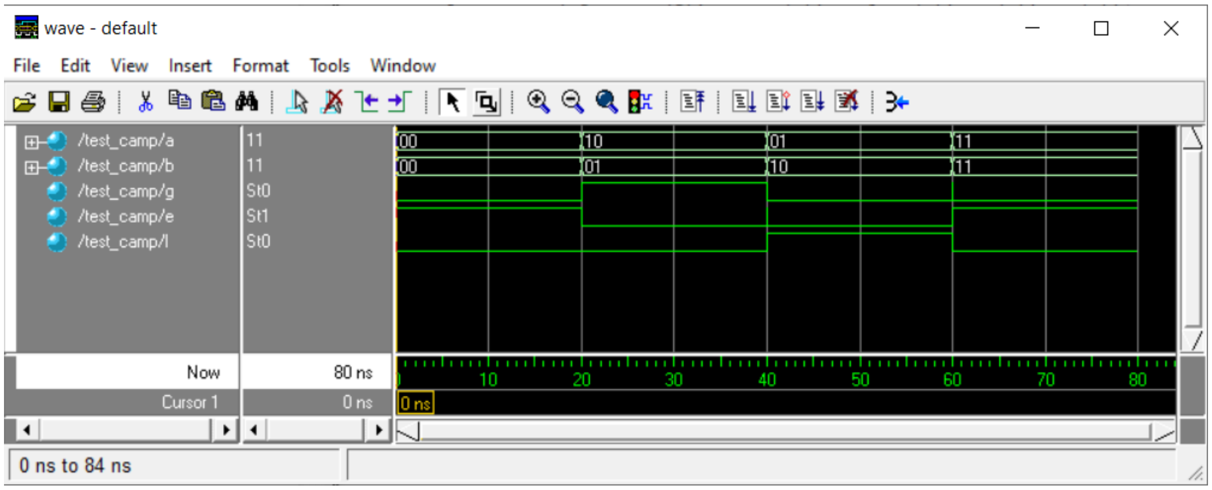
```
module camp2bit (greater, less, equal, A, B);

input[1:0] A, B;

output greater, less, equal;

wire an0, an1, bn0, bn1, w1, w2, w3, w4, w5, w6, w7, w8;

not n1(an0, A[0]);

not n2(an1, A[1]);

not n3(bn0, B[0]);

not n4(bn1, B[1]);

and a1(w1, A[0], bn0, bn1);

and a2(w2, A[1], bn1);

and a3(w3, A[1], A[0], bn0);

or o1(greater, w1, w2, w3);

xnor x1(w4, A[1], B[1]);
xnor x2(w5, A[0], B[0]);

and a4(equal, w4, w5);

and a5(w6, an1, B[1]);

and a6(w7, an0, B[1], B[0]);

and a7(w8, an1, an0, B[0]);

or o3(less, w6, w7, w8);

endmodule
```

Testbench for 2-bit Comparator

```
module test_camp();

reg[1:0] a, b;

wire g, e, l;

camp2bit CA1(.greater(g), .less(l), .equal(e), .A(a), .B(b));

initial

begin

a=2'b00;

b=2'b00;

#20

a=2'b10;

b=2'b01;

#20

a=2'b01;

b=2'b10;

#20

a=2'b11;

b=2'b11;

#20;

end
```

```
endmodule
```

Simulation Results



4-bit Comparator

```
module comp_2bit (greater2, equal2, less2, A2, B2);

input[1:0] A2, B2;

output reg greater2, equal2, less2;


always@(A2 or B2)


if (A2 > B2) begin

greater2=1;

equal2=0;

less2=0;

end


else if (A2 == B2) begin

greater2=0;

equal2=1;

less2=0;
```

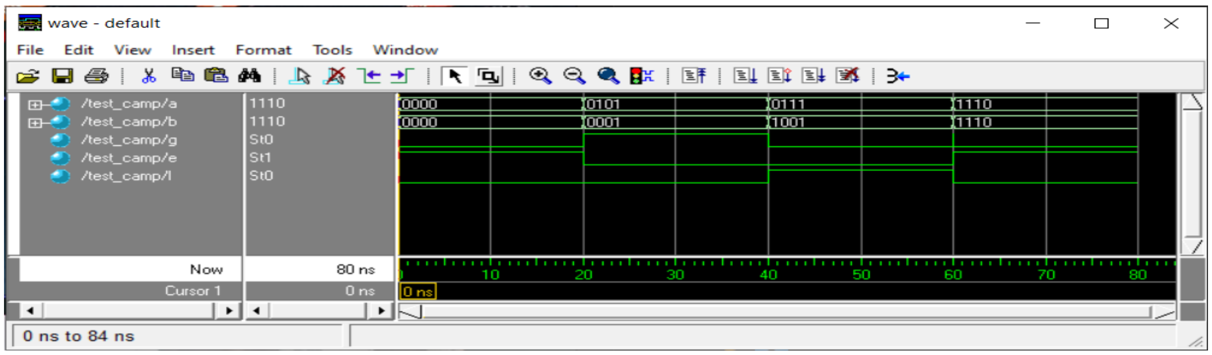
```
end

else begin
greater2=0;
equal2=0;
less2=1;
end
endmodule

module comp_4bit (greater, equal, less, A, B);
input[3:0] A, B;
output greater, equal, less;
wire[5:0] w;
wire w1, w2;
comp_2bit C1(.greater2(w[0]), .equal2(w[1]), .less2(w[2]), .A2(A[1:0]), .B2(B[1:0]));
comp_2bit C2(.greater2(w[3]), .equal2(w[4]), .less2(w[5]), .A2(A[3:2]), .B2(B[3:2]));

assign w1= w[4] & w[0];
assign w2= w[4] & w[2];
assign equal= w[4] & w[1];
assign greater= w[3] | w1;
assign less= w[5] | w2;

endmodule
```

Testbench for 4-bit Comparator

```
module test_camp();
```

```
reg[3:0] a, b;

wire g, e, l;

comp_4bit C1(.greater(g), .less(l), .equal(e), .A(a), .B(b));

initial

begin

a=4'b0000;

b=4'b0000;

#20

a=4'b0101;

b=4'b0001;

#20

a=4'b0111;

b=4'b1001;

#20

a=4'b1110;

b=4'b1110;

#20;

end

endmodule
```

Simulation Results



4.7 Observations

The testbenches for both the comparators contain 4 different inputs with a delay of 20 time units which can be clearly seen on the simulations. The outputs are as expected which means that the codes are working fine. The dataflow abstraction is much simpler to implement as compared to gate level coding. For the 4 bit comparator, we first implemented the 2-bit comparator and then used 2 of those comparators to implement a 4-bit comparator. This could also have been simply done by writing the code for a 4-bit comparator using dataflow, the code would have been reduced further.

4.8 Results

Dataflow abstraction makes the codes shorter as compared to the gate level. Any size of the comparator can be implemented using a few lines of code. We can also use instances of smaller modules to implement bigger modules. For example, now to implement an 8-bit comparator we can use 2 instances of 4-bit comparators and so on.