Digital Design Lab Report

Name	Dev Goel
Roll No.	B20CS090
Experiment Number	**III**

Objective

(i) Modify the Four Bit Adder created in Lab 1 to implement an Adder-Subtractor Unit.
- **a.** Indicate overflow
- **b.** Modify the unit to implement a comparator.
- **c.** Test using the test benches.

(ii) Write a Program to Implement the following Fast Adders [32 bit].
- **a.** Carry Look Ahead Adder
- **b.** Carry Skip Adder
- **c.** Carry Select Adder

NOTE: Implement the adders with full adder circuit having the following GATE Delays X-OR: 10 ns, AND, OR: 5 ns, MUX: 0 ns

So, I modified the codes of half adder and full adder in order to add the delay:

Half Adder (Code with delay for Fast Adders [32 bit] Implementation):

```
`timescale 1ns / 1ps
module half_adder (a, b, sum, carry);
input a, b;
output sum, carry;
assign #10 sum = a^b;
assign #5 carry = a&b;
endmodule
```

Full Adder (Code with delay for Fast Adders [32 bit] Implementation):

```
`timescale 1ns / 1ps
module full_adder (a, b, c, sum, carry);
input a, b, c;
output sum, carry;
wire ha1sum, ha1carry, ha2sum, ha2carry;
half_adder ha1(b, c, ha1sum, ha1carry);
half_adder ha2(a, ha1sum, ha2sum, ha2carry);
assign sum = ha2sum;
assign #5 carry = ha1carry|ha2carry;
endmodule
```

EXPERIMENT (i): Modify the Four Bit Adder created in Lab 1 to implement an Adder-Subtractor Unit.

- Indicate overflow

Adder-Subtractor Unit (Code):

```
`timescale 1ns / 1ps
module adder_subtractor(inp_A, inp_B, inp_M, out_C, out_S, overflow);
input [3:0] inp_A, inp_B;
input inp_M;
output [3:0] out_S;
output out_C, overflow;
wire [3:0] inp_Bi;
wire out_C1, out_C2, out_C3, out_C4;
xor_gate g1(inp_M, inp_B[0], inp_Bi[0]);
xor_gate g2(inp_M, inp_B[1], inp_Bi[1]);
xor_gate g3(inp_M, inp_B[2], inp_Bi[2]);
xor_gate g4(inp_M, inp_B[3], inp_Bi[3]);
full_adder fa1(inp_A[0], inp_Bi[0], inp_M, out_S[0], out_C1);
full_adder fa2(inp_A[1], inp_Bi[1], out_C1, out_S[1], out_C2);
full_adder fa3(inp_A[2], inp_Bi[2], out_C2, out_S[2], out_C3);
full_adder fa4(inp_A[3], inp_Bi[3], out_C3, out_S[3], out_C4);
xor_gate g5(out_C3, out_C4, overflow); //overflow
assign out_C = out_C4;
endmodule
```

Test Bench:

```
`timescale 1ns / 1ps
module test_adder_subtractor;
reg [3:0] inp_A, inp_B;
reg inp_M;
wire [3:0] out_S;
wire out_C, overflow;
adder_subtractor addsub(inp_A, inp_B, inp_M, out_C, out_S,overflow);
initial
   begin
      inp_M = 0; inp_A = 0; inp_B = 0;
      #5 inp_A = 4'b1100; inp_B = 4'b1011;
      #5 inp_A = 4'b0101; inp_B = 4'b1001;
      #5 inp_A = 4'b1010; inp_B = 4'b0001;
      #5 inp_M = 1; inp_A = 4'b0100; inp_B = 4'b0010;
      #5 inp_A = 4'b1010; inp_B = 4'b0110;
     end
initial #30 $finish;
endmodule
```

Waveform:

- Modify the unit to implement a comparator.

Comparator (Code):

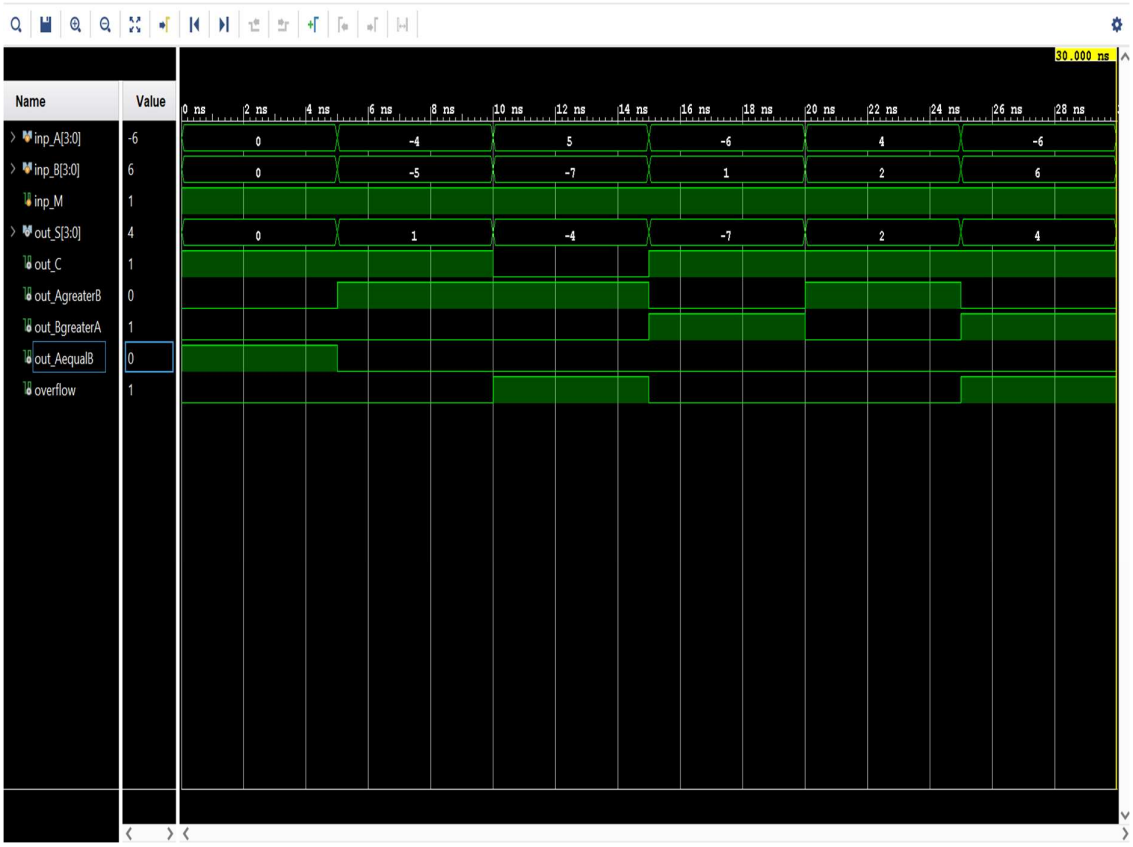
```
`timescale 1ns / 1ps
module comparator (inp_A, inp_B, inp_M, out_S, out_C, overflow,
out_AequalB, out_AgreaterB, out_BgreaterA);
input [3:0] inp_A, inp_B;
input inp_M;
output [3:0] out_S;
output  out_C, out_AgreaterB, out_BgreaterA, out_AequalB, overflow;
wire temp;
assign inp_M = 1;
adder_subtractor addsub(inp_A, inp_B, inp_M,out_C, out_S, overflow);
assign out_AequalB = (inp_A == inp_B)? 1: 0;
xnor (temp,overflow,out_S[3]);
assign out_AgreaterB = (out_AequalB == 0 & temp == 1)? 1:0;
xor(out_BgreaterA,overflow,out_S[3]);
endmodule
```

Test Bench:

```
`timescale 1ns / 1ps
module test_comparator;
reg [3:0] inp_A, inp_B;
reg inp_M;
wire [3:0] out_S;
wire out_C, out_AgreaterB, out_BgreaterA, out_AequalB, overflow;
comparator cmp(inp_A, inp_B, inp_M, out_S, out_C, overflow, out_AequalB,
out_AgreaterB, out_BgreaterA);
initial
   begin
      inp_M = 1; inp_A = 0; inp_B = 0;
      #5 inp_A = 4'b1100; inp_B = 4'b1011;
      #5 inp_A = 4'b0101; inp_B = 4'b1001;
      #5 inp_A = 4'b1010; inp_B = 4'b0001;
      #5 inp_A = 4'b0100; inp_B = 4'b0010;
      #5 inp_A = 4'b1010; inp_B = 4'b0110;
```

```
    end
initial #30 $finish;
endmodule
```

Waveform:



EXPERIMENT (ii): Write a Program to Implement the following Fast Adders

a. Carry Look Ahead Adder

```
`timescale 1ns / 1ps
module carry_look_ahead_adder(inp_A, inp_B, inp_cin, out_S, out_cout);

input [31:0]inp_A,inp_B;
input inp_cin;
output [31:0]out_S;
output out_cout;
wire [7:1]c;
wire [31:0]p,q;

fourbit_carry_look_ahead_adder fbcla1(inp_A[3:0], inp_B[3:0], inp_cin,
out_S[3:0], c[1], p[3:0], q[3:0]);
fourbit_carry_look_ahead_adder fbcla2(inp_A[7:4], inp_B[7:4], c[1],
out_S[7:4], c[2], p[7:4], q[7:4]);
fourbit_carry_look_ahead_adder fbcla3(inp_A[11:8], inp_B[11:8], c[2],
out_S[11:8], c[3], p[11:8], q[11:8]);
fourbit_carry_look_ahead_adder fbcla4(inp_A[15:12], inp_B[15:12], c[3],
out_S[15:12], c[4], p[15:12], q[15:12]);
fourbit_carry_look_ahead_adder fbcla5(inp_A[19:16], inp_B[19:16], c[4],
out_S[19:16], c[5], p[19:16], q[19:16]);
fourbit_carry_look_ahead_adder fbcla6(inp_A[23:20], inp_B[23:20], c[5],
out_S[23:20], c[6], p[23:20], q[23:20]);
fourbit_carry_look_ahead_adder fbcla7(inp_A[27:24], inp_B[27:24], c[6],
out_S[27:24], c[7], p[27:24], q[27:24]);
fourbit_carry_look_ahead_adder fbcla8(inp_A[31:28], inp_B[31:28], c[7],
out_S[31:28], out_cout, p[31:28], q[31:28]);

endmodule
```

- For carry look ahead adder implementation I wrote the code for implementing four bits carry ahead adder circuit first i.e.,

```
`timescale 1ns / 1ps
module fourbit_carry_look_ahead_adder(inp_A, inp_B, inp_cin, out_S,
out_cout, p, g);
input [3:0] inp_A, inp_B;
input inp_cin;
output [3:0] out_S, p, g;
output out_cout;
wire [3:1]c;
wire [10:1]aw;
wire [6:1]ow;
half_adder ha1(inp_A[0], inp_B[0], p[0], g[0]);
half_adder ha2(inp_A[1], inp_B[1], p[1], g[1]);
half_adder ha3(inp_A[2], inp_B[2], p[2], g[2]);
half_adder ha4(inp_A[3], inp_B[3], p[3], g[3]);
// Logic
// c0 = cin (input carry)
// c1 = g[0] + p[0]c[0]
// c2 = g[1] + p[1]g[0] + p[1]p[0]c[0]
// c3 = g[2] + p[2]g[1] + p[2]p[1]g[0] + p[2]p[2]p[0]c[0]
and #5 A1(aw[1], p[0], inp_cin);
or #5 O1(c[1], aw[1], g[0]);
and #5 A2(aw[2], p[1], g[0]);
and #5 A3(aw[3], p[1], aw[1]);
or #5 O2(ow[1], aw[3], aw[2]);
or #5 O3(c[2], ow[1], g[1]);
and #5 A4(aw[4], p[2], g[1]);
and #5 A5(aw[5], aw[2], p[2]);
and #5 A6(aw[6], p[2], aw[3]);
or #5 O4(ow[2], aw[5], aw[6]);
or #5 O5(ow[3], aw[4], ow[2]);
or #5 O6(c[3], g[2], ow[3]);
and #5 A7(aw[7], p[3], g[2]);
and #5 A8(aw[8], p[3], aw[4]);
```

```
and #5 A9(aw[9], p[3], aw[5]);
and #5 A10(aw[10], p[3], aw[6]);
or #5 O7(ow[4], aw[10], aw[9]);
or #5 O8(ow[5], ow[4], aw[8]);
or #5 O9(ow[6], ow[5], aw[7]);
or #5 O10(out_cout, ow[6], g[3]);
xor #10 x1(out_S[0], p[0], inp_cin);
xor #10 x2(out_S[1], p[1], c[1]);
xor #10 x3(out_S[2], p[2], c[2]);
xor #10 x4(out_S[3], p[3], c[3]);
endmodule
```

Test Bench:

```
`timescale 1ns / 1ps
module test_carry_look_ahead_adder;
reg [31:0] inp_A, inp_B;
reg inp_cin;
wire [31:0] out_S;
wire out_cout;
integer i, j;
carry_look_ahead_adder cla1(inp_A, inp_B, inp_cin, out_S, out_cout);
initial
   begin
      for(i = 0 ; i<8; i=i+1)
         begin
            for(j = 0 ; j<8 ; j=j+1)
               begin
                  assign inp_cin = 0;
                  assign inp_A = i;
                  assign inp_B = j; #10;
               end
         end
   end
endmodule
```

Waveform:

- **With Delay in Gates:**

- **Without Delay in Gates:**

b. Carry Skip Adder

```
`timescale 1ns / 1ps
module carry_skip_adder(inp_A, inp_B, inp_cin, out_S, out_cout);
input [31:0] inp_A, inp_B;
input inp_cin;
output [31:0] out_S;
output out_cout;
wire [7:1]c;
four_bit_carry_skip_adder fbcsa1(inp_A[3:0], inp_B[3:0], inp_cin,
out_S[3:0], c[1]);
four_bit_carry_skip_adder fbcsa2(inp_A[7:4], inp_B[7:4], c[1],
out_S[7:4], c[2]);
four_bit_carry_skip_adder fbcsa3(inp_A[11:8], inp_B[11:8], c[2],
out_S[11:8], c[3]);
four_bit_carry_skip_adder fbcsa4(inp_A[15:12], inp_B[15:12], c[3],
out_S[15:12], c[4]);
four_bit_carry_skip_adder fbcsa5(inp_A[19:16], inp_B[19:16], c[4],
out_S[19:16], c[5]);
four_bit_carry_skip_adder fbcsa6(inp_A[23:20], inp_B[23:20], c[5],
out_S[23:20], c[6]);
four_bit_carry_skip_adder fbcsa7(inp_A[27:24], inp_B[27:24], c[6],
out_S[27:24], c[7]);
four_bit_carry_skip_adder fbcsa8(inp_A[31:28], inp_B[31:28], c[7],
out_S[31:28], out_cout);
endmodule
```

- For carry skip adder implementation, I wrote the code for implementing four bits carry skip adder circuit first i.e.,

```
`timescale 1ns / 1ps
module four_bit_carry_skip_adder(inp_A, inp_B, inp_cin, out_S,
out_cout);
input [3:0] inp_A, inp_B;
input inp_cin;
output [3:0] out_S;
output out_cout;
```

```
wire [3:0]c;
wire [3:0]p;
wire [3:1]aw;
full_adder fa1(inp_A[0], inp_B[0], inp_cin, out_S[0], c[0]);
full_adder fa2(inp_A[1], inp_B[1], c[0], out_S[1], c[1]);
full_adder fa3(inp_A[2], inp_B[2], c[1], out_S[2], c[2]);
full_adder fa4(inp_A[3], inp_B[3], c[2], out_S[3], c[3]);
assign #10 p[0] = inp_A[0]^inp_B[0];
assign #10 p[1] = inp_A[1]^inp_B[1];
assign #10 p[2] = inp_A[2]^inp_B[2];
assign #10 p[3] = inp_A[3]^inp_B[3];
and #5 A1(aw[1], p[0], p[1]);
and #5 A2(aw[2], p[2], p[3]);
and #5 A3(aw[3], aw[1], aw[2]);
multiplexer mux1(c[3], inp_cin ,aw[3], out_cout);
endmodule
```

- Also, Multiplexer Code was used in the implementation.

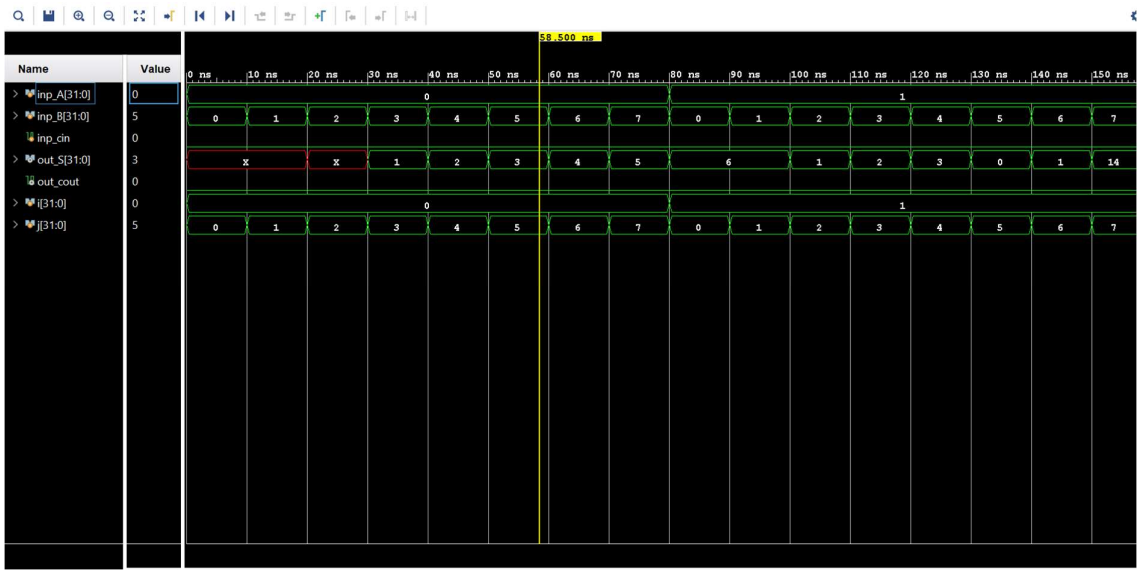
```
`timescale 1ns / 1ps
module multiplexer(i0,i1,sel,bitout);
input i0,i1, sel;
output reg bitout;
always@(i0,i1,sel)
   begin
      if(sel == 0)
         bitout = i0;
      else
         bitout = i1;
   end
endmodule
```

Test Bench:

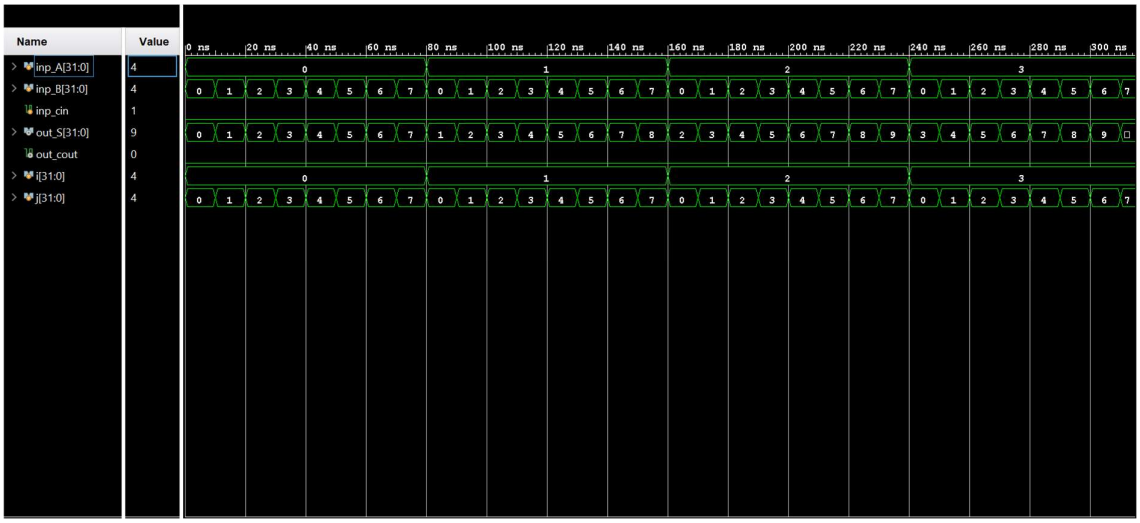
```
`timescale 1ns / 1ps
module test_carry_skip_adder;

reg [31:0] inp_A, inp_B;
reg inp_cin;
wire [31:0] out_S;
wire out_cout;
integer i,j;
carry_skip_adder csa1(inp_A, inp_B, inp_cin, out_S, out_cout);
initial
   begin
      assign inp_cin = 0;
      for(i = 0; i<8; i=i+1)
         begin
            for(j = 0; j<8 ; j=j+1)
               begin
                  assign inp_A = i;
                  assign inp_B = j; #10;
               end
         end
      assign inp_cin = 1;
      for(i = 0; i<8; i=i+1)
         begin
            for(j = 0; j<8 ; j=j+1)
               begin
                  assign inp_A = i;
                  assign inp_B = j; #10;
               end
         end
   end
endmodule
```

- **Waveform:**

- **With delay in gates.**



- **Without delay in gates.**



c. Carry Select Adder

```
`timescale 1ns / 1ps
module CarrySelectAdder(A, B, cin, S, cout);
input [31:0] A, B;
input cin;
output [31:0] S;
output cout;
wire [31:0] temp0, temp1, carry0, carry1;
genvar i;
full_adder fa00(A[0], B[0], 1'b0, temp0[0], carry0[0]);
for(i = 1;i < 32 ; i=i+1)
   begin
      full_adder fa01(A[i], B[i], carry0[i-1], temp0[i], carry0[i]);
   end
full_adder fa10(A[0], B[0], 1'b1, temp1[0], carry1[0]);
   for(i = 1;i < 32 ; i=i+1)
      begin
         full_adder fa11(A[i], B[i], carry1[i-1], temp1[i], carry1[i]);
      end
multiplexer mux_carry(carry0[31], carry1[31], cin, cout);
for(i = 0; i < 32 ; i=i+1)
   begin
      multiplexer mux_sum(temp0[i], temp1[i], cin, S[i]);
   end
endmodule
```

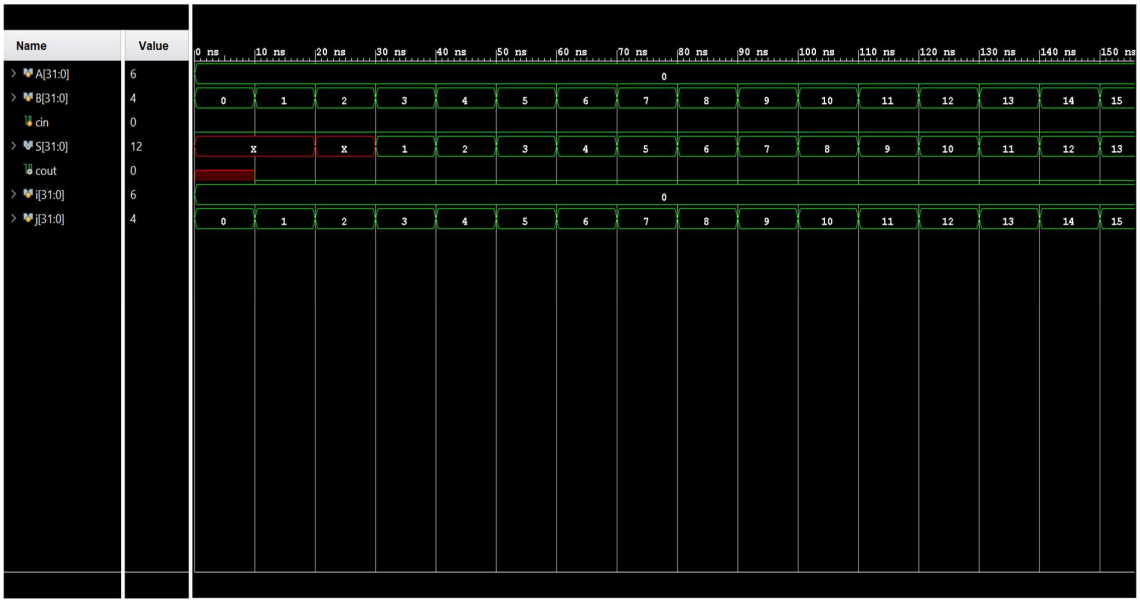
Test Bench:

```
`timescale 1ns / 1ps
module Test_CarrySelectAdder;
reg [31:0] A, B;
reg cin;
wire [31:0] S;
```

```
wire cout;
integer i, j;

CarrySelectAdder csa(A, B, cin, S, cout);

initial
   begin
      A = 0;
      B = 0;
      cin = 0;
      for(i=0;i<16;i=i+1)
         begin
            for(j=0;j<16;j=j+1)
               begin
                  A = i;
                  B = j;
                  #10;
               end
         end

      cin = 1;
      for(i=0;i<16;i=i+1)
         begin
            for(j=0;j<16;j=j+1)
               begin
                  A = i;
                  B = j;
                  #10;
               end
         end
   end
endmodule
```

- **Waveform:**

- **With delay in Gates:**



- **Without delay in Gates:**