DIGITAL DESIGN

LAB-3 REPORT

1. Adder-Subtractor:

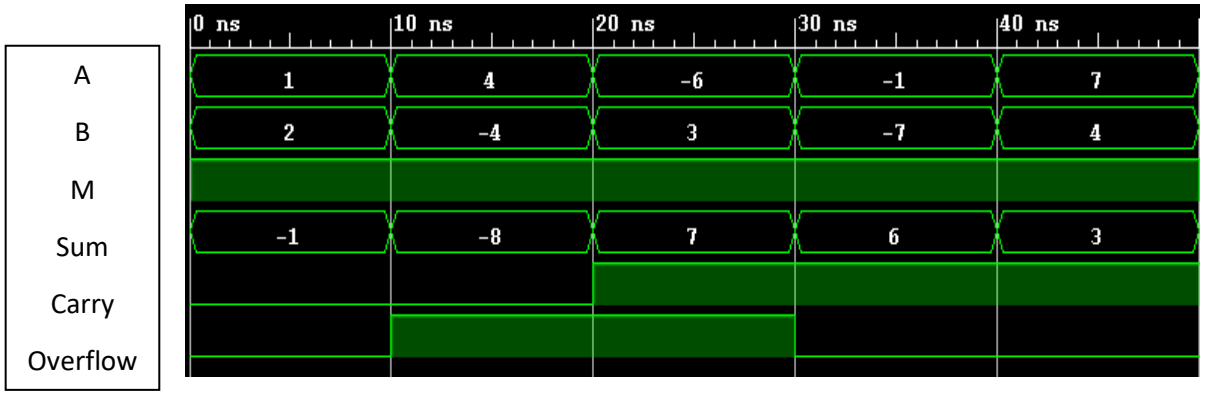
Logic Code:

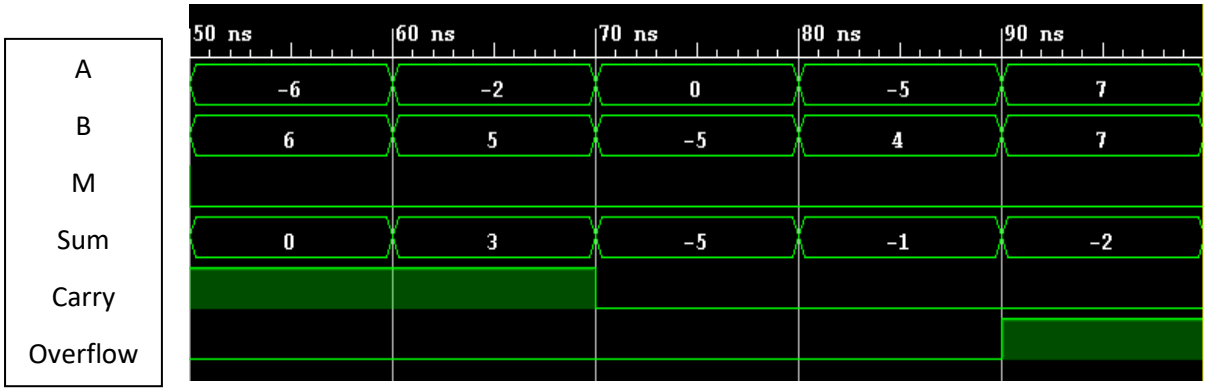
```
module half_adder(A,B,sum,carry);...

module full_adder(A,B,C,sum,carry);...

module adder_subtractor(A,B,M,S,carry,V);
input [3:0]A,B;
input M;
output [3:0]S;
output carry,V;
wire C0,C1,C2,C3;

assign C0=M;
full_adder FA1(A[0],B[0]^M,C0,S[0],C1);
full_adder FA2(A[1],B[1]^M,C1,S[1],C2);
full_adder FA3(A[2],B[2]^M,C2,S[2],C3);
full_adder FA4(A[3],B[3]^M,C3,S[3],carry);
assign V=carry^C3;
endmodule
```

Test Bench:

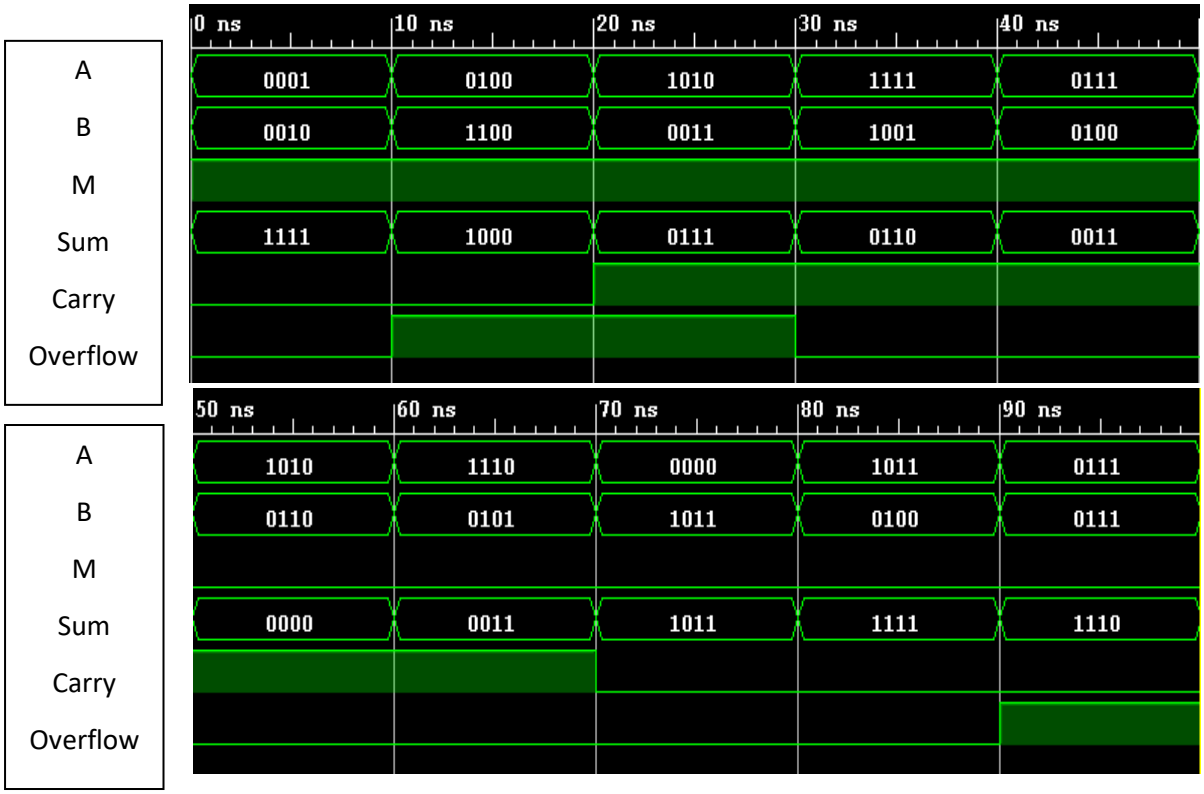
```
module test_adder_subtractor();
reg [3:0]A,B;
reg M;
wire [3:0]S;
wire carry,overflow;
adder_subtractor AS1(A,B,M,S,carry,overflow);
initial
begin
    A=4'b0001; B=4'b0010; M=1'b1;
    #10 A=4'b1100; B=4'b0100;
    #10 A=4'b0011; B=4'b1010;
    #10 A=4'b1111; B=4'b1001;
    #10 A=4'b0111; B=4'b0100;
    #10 A=4'b1010; B=4'b0110; M=1'b0;
    #10 A=4'b1110; B=4'b0101;
    #10 A=4'b0000; B=4'b1011;
    #10 A=4'b1011; B=4'b0100;
    #10 A=4'b0111; B=4'b0111;
end
initial #100 $finish;
endmodule
```

Decimal Form:





Binary Form:



2. Comparator:

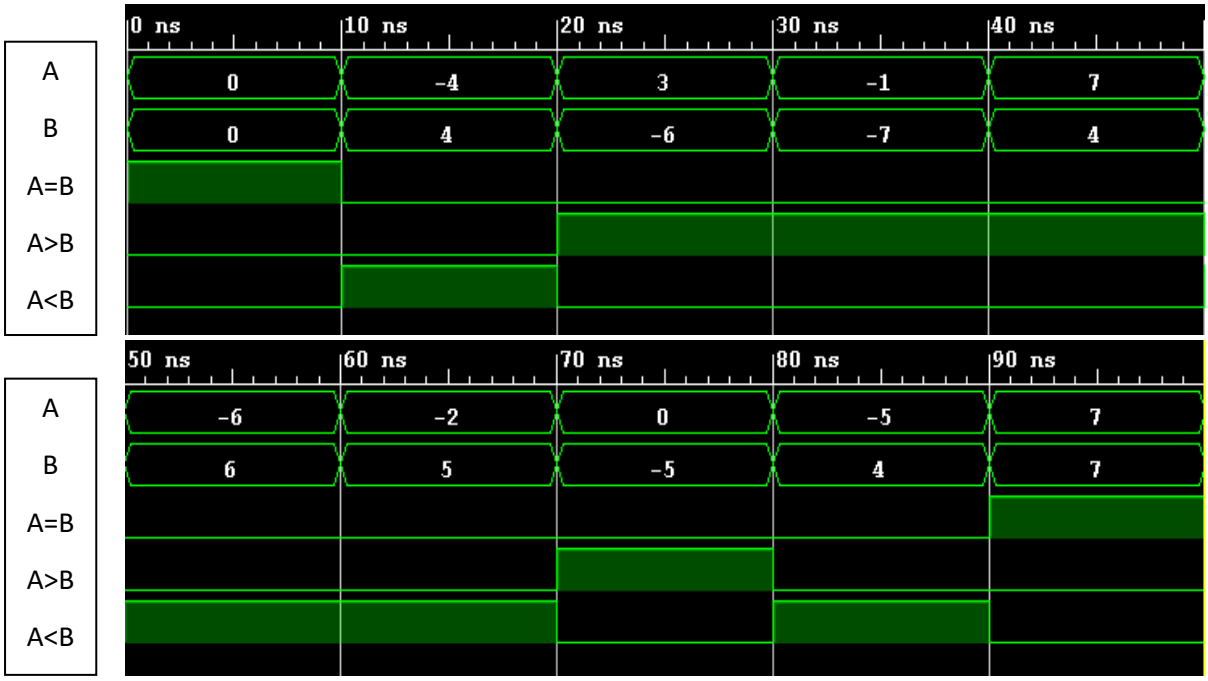
Logic Code:

```
module comparator(A,B,E,X,Y);
input [3:0] A,B;
output E,X,Y;
reg M=1'b1;
wire [3:0] S;
wire C,V;

adder_subtractor AS2(A,B,M,S,C,V);
assign Y=V^S[3];
assign E= ~(S[3]|S[2]|S[1]|S[0]);
assign X=~(Y|E);
endmodule
```

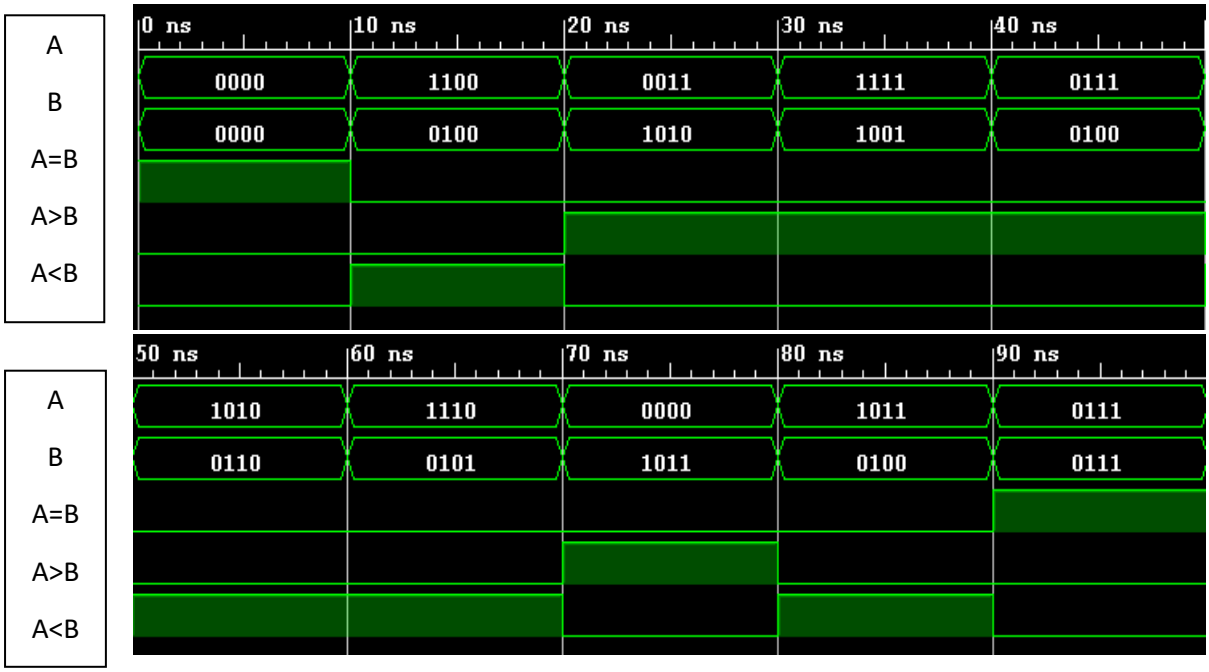
Test Bench:

```
module test_comparator();
reg [3:0] A,B;
wire Equality,A_greater,B_greater;
comparator C1(A,B,Equality,A_greater,B_greater);

initial
begin
    A=4'b0000; B=4'b0000;
    #10 A=4'b1100; B=4'b0100;
    #10 A=4'b0011; B=4'b1010;
    #10 A=4'b1111; B=4'b1001;
    #10 A=4'b0111; B=4'b0100;
    #10 A=4'b1010; B=4'b0110;
    #10 A=4'b1110; B=4'b0101;
    #10 A=4'b0000; B=4'b1011;
    #10 A=4'b1011; B=4'b0100;
    #10 A=4'b0111; B=4'b0111;
end
initial #100 $finish;
endmodule
```

Decimal Form:



Binary Form:



3. Fast Adders:

a. Carry Look Ahead Adder:

Logic Code:

```
module four_bit_adder(A,B,C0,S,C_out);
input [3:0] A,B;
input C0;
output [3:0] S;
output C_out;
wire [3:0] P,C,G;

assign #10 P[0]=A[0]^B[0]; assign #10 P[1]=A[1]^B[1]; assign #10 P[2]=A[2]^B[2]; assign #10 P[3]=A[3]^B[3];
assign #5 G[0]=A[0]&B[0]; assign #5 G[1]=A[1]&B[1]; assign #5 G[2]=A[2]&B[2]; assign #5 G[3]=A[3]&B[3];
assign C[0]=C0;
assign #10 C[1]=G[0]|(P[0]&C[0]);
assign #10 C[2]=G[1]|(P[1]&G[0])|(P[1]&P[0]&C[0]);
assign #10 C[3]=G[2]|(P[2]&G[1])|(P[2]&P[1]&G[0])|(P[2]&P[1]&P[0]&C[0]);
assign #10 C_out=G[3]|(P[3]&G[2])|(P[3]&P[2]&G[1])|(P[3]&P[2]&P[1]&G[0])|(P[3]&P[2]&P[1]&P[0]&C[0]);
assign #10 S[0]=P[0]^C[0];
assign #10 S[1]=P[1]^C[1];
assign #10 S[2]=P[2]^C[2];
assign #10 S[3]=P[3]^C[3];
endmodule
```

```
module carry_look_ahead_adder(A,B,S,C);
input [31:0] A,B;
output [31:0] S;
output C;
reg C0=1'b0;
wire C1,C2,C3,C4,C5,C6,C7;

four_bit_adder FBA1(A[3:0],B[3:0],C0,S[3:0],C1);
four_bit_adder FBA2(A[7:4],B[7:4],C1,S[7:4],C2);
four_bit_adder FBA3(A[11:8],B[11:8],C2,S[11:8],C3);
four_bit_adder FBA4(A[15:12],B[15:12],C3,S[15:12],C4);
four_bit_adder FBA5(A[19:16],B[19:16],C4,S[19:16],C5);
four_bit_adder FBA6(A[23:20],B[23:20],C5,S[23:20],C6);
four_bit_adder FBA7(A[27:24],B[27:24],C6,S[27:24],C7);
four_bit_adder FBA8(A[31:28],B[31:28],C7,S[31:28],C);
endmodule
```

b. Carry Select Adder:

Logic Code:

```
module mux(X,Y,S,O);
input X,Y,S;
output O;

assign O=S?Y:X;
endmodule

module full_adders(A,B,C,S,C_out);
input A,B,C;
output S,C_out;
wire S0,C1,C2;

assign #10 S0=A^B;
assign #5 C1=A&B;
assign #10 S=S0^C;
assign #5 C2=S0&C;
assign #5 C_out=C2|C1;
endmodule

module four_bits_adder(A,B,C0,S,carry);
input [3:0]A,B;
input C0;
output [3:0]S;
output carry;
wire C1,C2,C3;

full_adders FAa(A[0],B[0],C0,S[0],C1);
full_adders FAb(A[1],B[1],C1,S[1],C2);
full_adders FAc(A[2],B[2],C2,S[2],C3);
full_adders FAd(A[3],B[3],C3,S[3],carry);
endmodule
```

```
module carry_select_adder(A,B,S,C_out);
input [31:0] A,B;
output [31:0] S;
output C_out;
reg Ca=1'b0,Cb=1'b1;
wire C8a,C2a,C3a,C4a,C5a,C6a,C7a,C8b,C2b,C3b,C4b,C5b,C6b,C7b,C1,C2,C3,C4,C5,C6,C7;
wire [31:4] S1,S2;

four_bits_adder FBAa(A[3:0],B[3:0],Ca,S[3:0],C1);
four_bits_adder FBAb(A[7:4],B[7:4],Ca,S1[7:4],C2a);
four_bits_adder FBAc(A[7:4],B[7:4],Cb,S2[7:4],C2b);
four_bits_adder FBAd(A[11:8],B[11:8],Ca,S1[11:8],C3a);
four_bits_adder FBAe(A[11:8],B[11:8],Cb,S2[11:8],C3b);
four_bits_adder FBAf(A[15:12],B[15:12],Ca,S1[15:12],C4a);
four_bits_adder FBAg(A[15:12],B[15:12],Cb,S2[15:12],C4b);
four_bits_adder FBAh(A[19:16],B[19:16],Ca,S1[19:16],C5a);
four_bits_adder FBAi(A[19:16],B[19:16],Cb,S2[19:16],C5b);
four_bits_adder FBAj(A[23:20],B[23:20],Ca,S1[23:20],C6a);
four_bits_adder FBAk(A[23:20],B[23:20],Cb,S2[23:20],C6b);
four_bits_adder FBAl(A[27:24],B[27:24],Ca,S1[27:24],C7a);
four_bits_adder FBAm(A[27:24],B[27:24],Cb,S2[27:24],C7b);
four_bits_adder FBAn(A[31:28],B[31:28],Ca,S1[31:28],C8a);
four_bits_adder FBAo(A[31:28],B[31:28],Cb,S2[31:28],C8b);
mux M1(C2a,C2b,C1,C2);
mux M2(C3a,C3b,C2,C3);
mux M3(C4a,C4b,C3,C4);
mux M4(C5a,C5b,C4,C5);
mux M5(C6a,C6b,C5,C6);
mux M6(C7a,C7b,C6,C7);
mux M7(C8a,C8b,C7,C_out);
mux Ma1(S1[4],S2[4],C1,S[4]); mux Mb1(S1[5],S2[5],C1,S[5]); mux Mc1(S1[6],S2[6],C1,S[6]); mux Md1(S1[7],S2[7],C1,S[7]);
mux Ma2(S1[8],S2[8],C2,S[8]); mux Mb2(S1[9],S2[9],C2,S[9]); mux Mc2(S1[10],S2[10],C2,S[10]); mux Md2(S1[11],S2[11],C2,S[11]);
mux Ma3(S1[12],S2[12],C3,S[12]); mux Mb3(S1[13],S2[13],C3,S[13]); mux Mc3(S1[14],S2[14],C3,S[14]); mux Md3(S1[15],S2[15],C3,S[15]);
mux Ma4(S1[16],S2[16],C4,S[16]); mux Mb4(S1[17],S2[17],C4,S[17]); mux Mc4(S1[18],S2[18],C4,S[18]); mux Md4(S1[19],S2[19],C4,S[19]);
mux Ma5(S1[20],S2[20],C5,S[20]); mux Mb5(S1[21],S2[21],C5,S[21]); mux Mc5(S1[22],S2[22],C5,S[22]); mux Md5(S1[23],S2[23],C5,S[23]);
mux Ma6(S1[24],S2[24],C6,S[24]); mux Mb6(S1[25],S2[25],C6,S[25]); mux Mc6(S1[26],S2[26],C6,S[26]); mux Md6(S1[27],S2[27],C6,S[27]);
mux Ma7(S1[28],S2[28],C7,S[28]); mux Mb7(S1[29],S2[29],C7,S[29]); mux Mc7(S1[30],S2[30],C7,S[30]); mux Md7(S1[31],S2[31],C7,S[31]);
endmodule
```

c. Carry Skip Adder:

```
module full_skip_adders(A,B,C,S,S0,C_out);
input A,B,C;
output S,S0,C_out;
wire C1,C2;

assign #10 S0=A^B;
assign #5 C1=A&B;
assign #10 S=S0^C;
assign #5 C2=S0&C;
assign #5 C_out=C2|C1;
endmodule

module skip_adder(A,B,C0,S,carry);
input [3:0]A,B;
input C0;
output [3:0]S;
output carry;
wire [3:0] P;
wire C1,C2,C3,C4,PS;


full_skip_adders FSAa(A[0],B[0],C0,S[0],P[0],C1);
full_skip_adders FSAb(A[1],B[1],C1,S[1],P[1],C2);
full_skip_adders FSAc(A[2],B[2],C2,S[2],P[2],C3);
full_skip_adders FSAd(A[3],B[3],C3,S[3],P[3],C4);
assign #5 PS=P[0]&P[1]&P[2]&P[3];
mux Ms(C4,C0,PS,carry);
endmodule
module carry_skip_adder(A,B,S,C);
input [31:0] A,B;
output [31:0] S;
output C;
reg C0=1'b0;
wire C1,C2,C3,C4,C5,C6,C7;

skip_adder SA1(A[3:0],B[3:0],C0,S[3:0],C1);
skip_adder SA2(A[7:4],B[7:4],C1,S[7:4],C2);
skip_adder SA3(A[11:8],B[11:8],C2,S[11:8],C3);
skip_adder SA4(A[15:12],B[15:12],C3,S[15:12],C4);
skip_adder SA5(A[19:16],B[19:16],C4,S[19:16],C5);
skip_adder SA6(A[23:20],B[23:20],C5,S[23:20],C6);
skip_adder SA7(A[27:24],B[27:24],C6,S[27:24],C7);
skip_adder SA8(A[31:28],B[31:28],C7,S[31:28],C);
endmodule
```

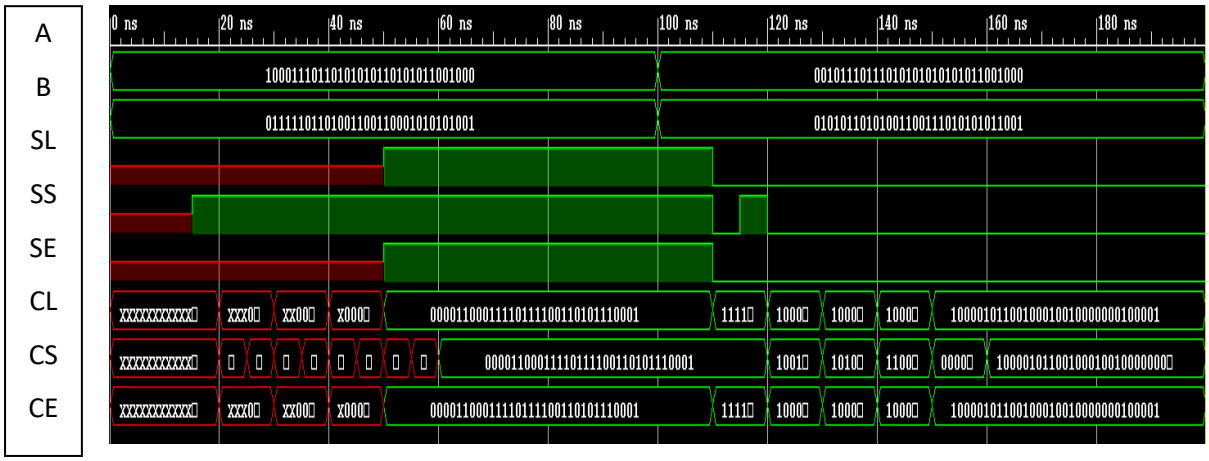
Test Bench:

```
module test_compare();
reg [31:0] A,B;
wire Carry_look_ahead,Carry_skip,Carry_select;
wire [31:0] S_look_ahead,S_skip,S_select;
carry_select_adder CSA1(A,B,S_look_ahead,Carry_look_ahead);
carry_skip_adder CSAb(A,B,S_skip,Carry_skip);
carry_select_adder CSAc(A,B,S_select,Carry_select);
initial
begin
    A=32'b10001110110101010110101011001000;
    B=32'b01111101101001100110001010101001;
    #100
    A=32'b00101110111010101010101011001000;
    B=32'b01010110101001100111010101011001;
end
initial #200 $finish;
endmodule
```

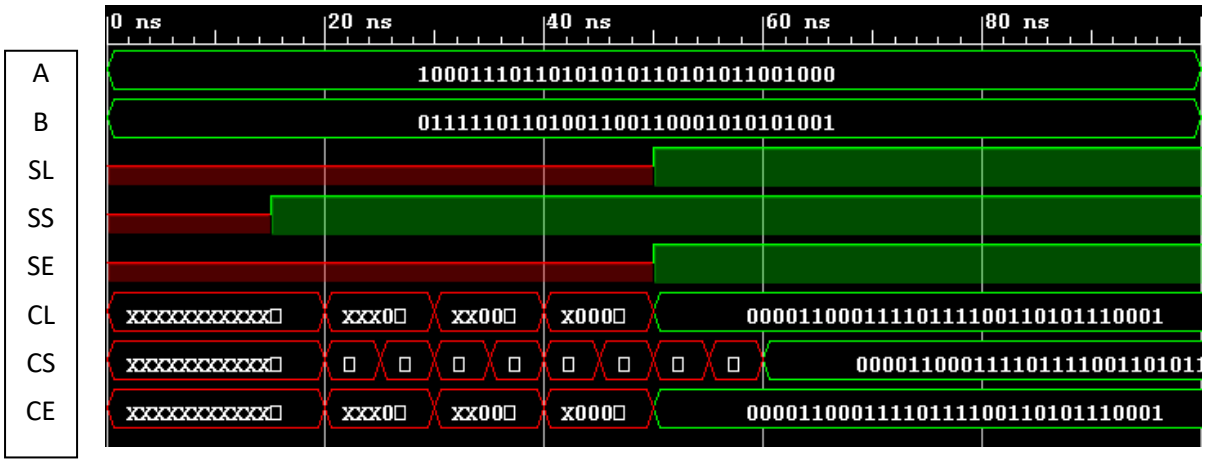
SL-Look Ahead Sum CL-Look Ahead Carry

SS-Skip Sum CS-Skip Carry

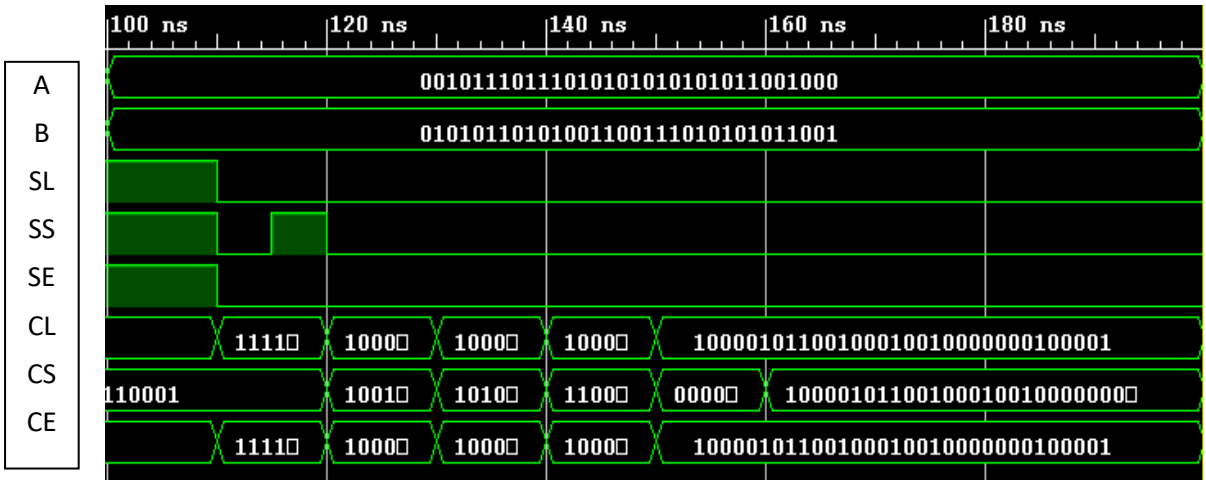
SE-Select Sum CE-Select Carry



Frist Case (initial 100 ns):



Second case (next 100 ns):



XOR gate delay: 10 ns

AND and OR gate delay: 5 ns

Multiplexer delay: 0 ns

Look Ahead Adder takes 50 ns in both cases to give the carry and sum.

Skip Adder takes 15 ns in the first case and 20 ns in the second case to give the carry but 60 ns for the sum in both cases.

Select Adder takes 50 ns in both cases to give the carry and sum.