Electrical and Computer Engineering Department

ENCS3310

ADVANCED DIGITAL SYSTEMS DESIGN

Project Report

Design a 2-digit BCD adder (8 bits)

Instructor: Dr. Abdallatif Abulssa.

Section: 1.

Student's Name: Saif Battah.

Student's ID: 1170986.

Introduction & Abstract

In this project we're going to create a 2-digit "BCD" adder.

We will use Verilog coding language using "Aldec" program to implement the code.

We will create this system using 1-bit full adder, 4-bit binary adder, 1-bit "BCD" adder and Registers.

So our program should work as "ADDING" Calculator, for example: if we tend to set **A = 54**, **B = 34**, so our program should add these numbers starting from Least significant bit which are **AL = 4, BL = 4** and give us a Least Sum which is **SL = 8**; and set carry to **0** or **1** depending whether the **SL** is less than or greater than **9** "IF **SL > 9 → set carry to 1, else carry = 0**", then after we finish the least significant we move to add the most significant with the carry from the least operation, which are

AM = 5, **BM = 3**, **Carry = 0**, so the Most significant sum is **8**.

After that we combine "not adding" the least sum with the most sum and final carry to get the adding result which is **88**.

And as I mentioned before I've built the system using these smaller system:

- Built 1-bit full adder from the basic gates.
- Use the full adder to build 4-bit adder.
- Use the 4-bit adders to build 1-Digit BCD Adder.
- Use the 1-Digit BCD Adder to build the 2-Digit BCD Adder

Also we have these delays to be used when we use a specific gate

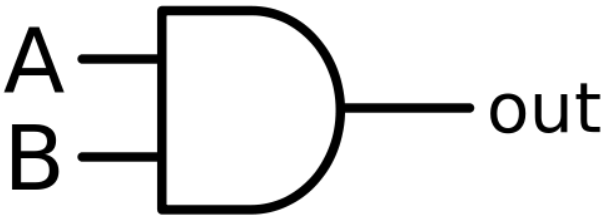
Gate	Delay
Inverter	3 ns
NAND	6 ns
NOR	6 ns
AND	8 ns
OR	8 ns
XNOR	10 ns
XOR	12 ns

Finally, after building the whole system we have to go over 2 stages, each stage has its own specification to test the output results and searching for any error in the adding process, so we will test 2^16 situations to check the validity and accuracy of our system.

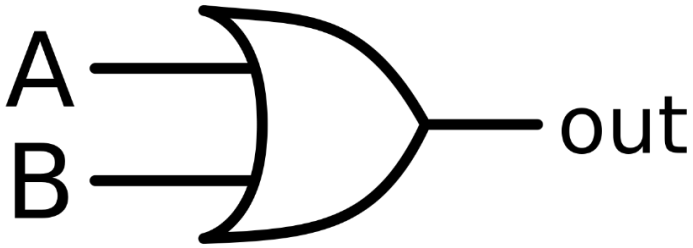
Program Design

Starting with basic gates:

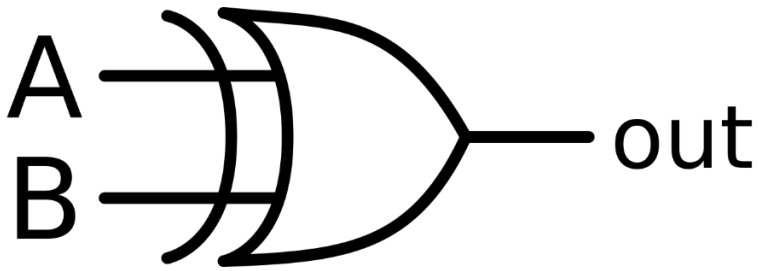
1. And Gate



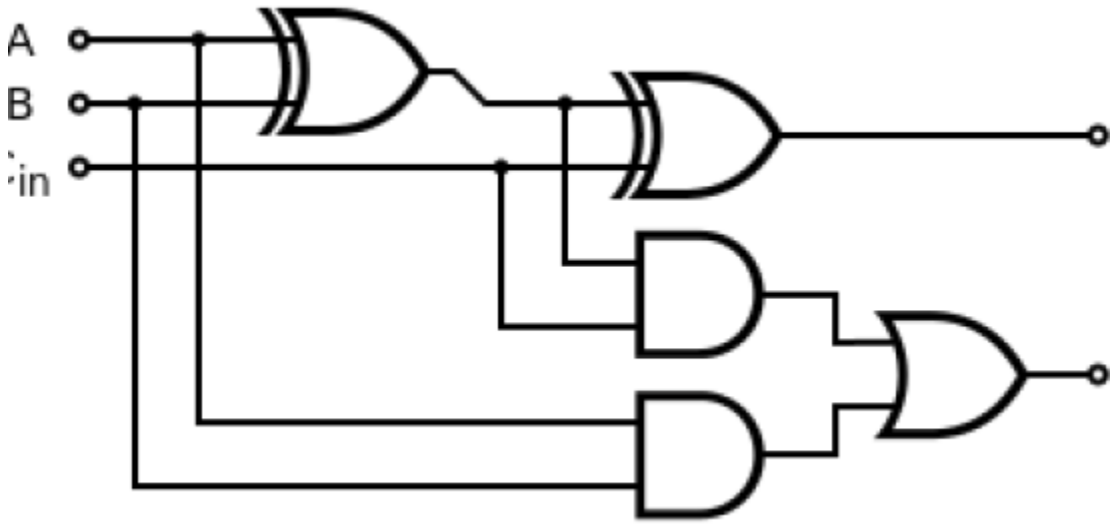
2. Or Gate



3. XOR Gate



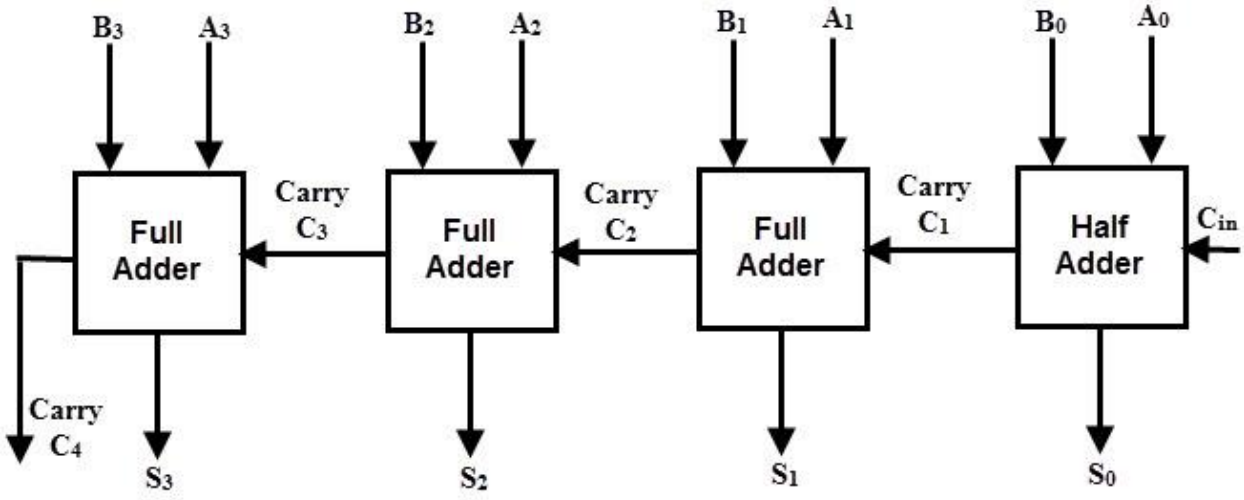
Full Adder (1-bit)



Code

```
module FA_1bit(A,B,Cin,Sum,Cout);

    input A,B,Cin;
    output Sum,Cout;
    wire w1,w2,w3;

    and #(8ns) (w1,A,B);
    xor #(12ns) (w2,A,B);
    and #(8ns) (w3,w2,Cin);
    xor #(12ns) (Sum,w2,Cin);
    or  #(8ns) (Cout,w1,w3);

endmodule
```

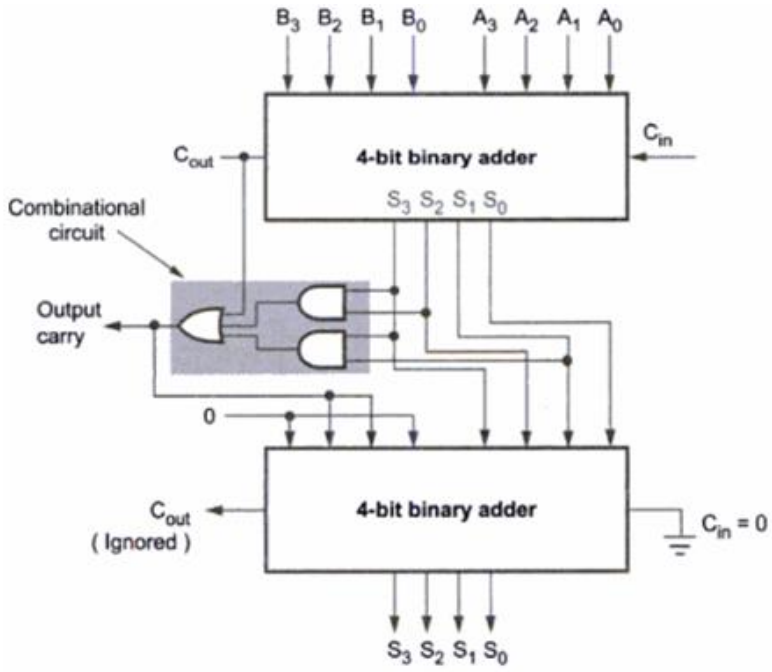
Binary Adder (4-bit)



Code

```
module BinaryAdder_4bit(A,B,Cin,Sum,Cout);

    input [3:0] A,B;
    input Cin;
    output [3:0] Sum;
    output Cout;
    wire C1,C2,C3;

    FA_1bit F1(A[0],B[0],Cin,Sum[0],C1);
    FA_1bit F2(A[1],B[1],C1,Sum[1],C2);
    FA_1bit F3(A[2],B[2],C2,Sum[2],C3);
    FA_1bit F4(A[3],B[3],C3,Sum[3],Cout);


endmodule
```

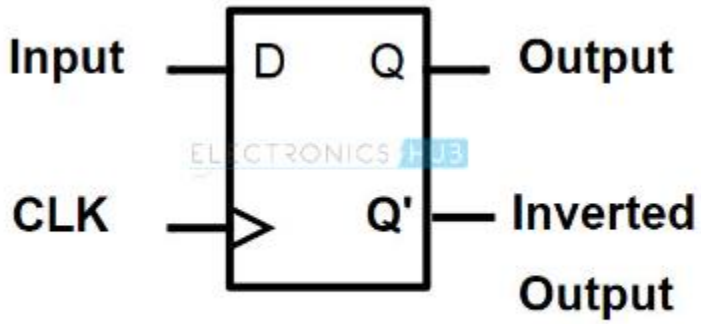
BCD Adder (1-bit)



Code

```
module BCD_Adder1bit(A,B,Cin,F,OutputCarry);
    input [3:0] A,B;
    input Cin;
    output [3:0] F;
    output OutputCarry;
    wire [3:0] Z,S;
    wire k,w1,w2,w3;

    BinaryAdder_4bit BA1(A,B,Cin,Z,k);

    and #(8ns) (w1,Z[2],Z[3]);
    and #(8ns) (w2,Z[1],Z[3]);
    or  #(8ns) (OutputCarry,k,w1,w2);

    assign S = {1'b0,OutputCarry,OutputCarry,1'b0};
    BinaryAdder_4bit BA2(Z,S,0,F,w3);
endmodule
```

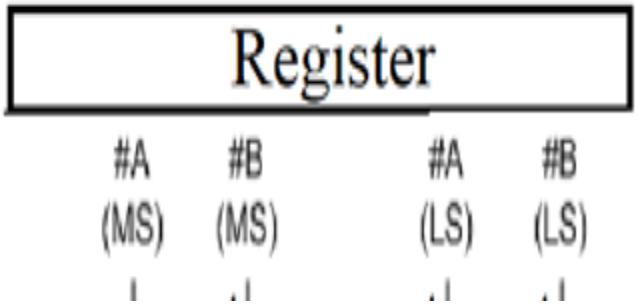
D Flip Flop



Code

```
module dff(Q,D,clk);
    input D,clk;
    output reg Q;

    always @(posedge clk)
        begin
            Q <= D;
        end
endmodule
```

Input Register



Code

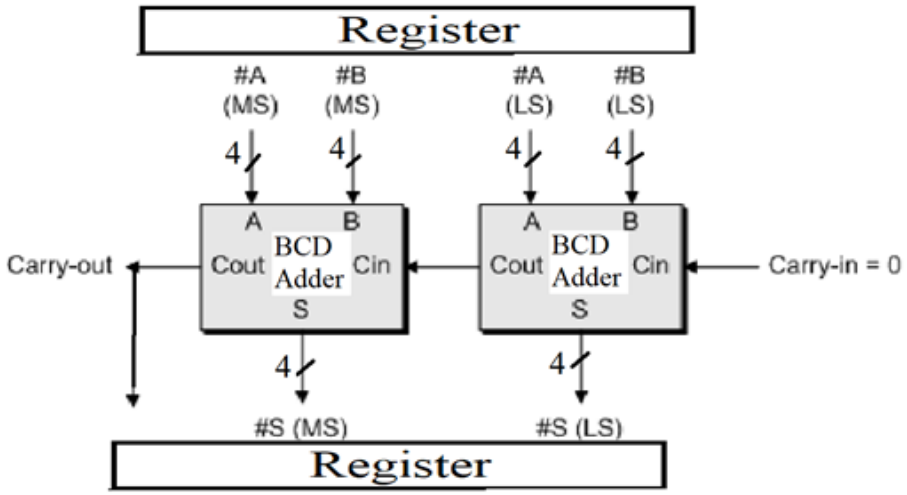
```
module Register_input(out,in,clk);
    output reg [7:0] out;
    input [7:0] in;
    input clk;
    wire [7:0] temp_out;

    dff d0(temp_out[0],in[0],clk);
    dff d1(temp_out[1],in[1],clk);
    dff d2(temp_out[2],in[2],clk);
    dff d3(temp_out[3],in[3],clk);
    dff d4(temp_out[4],in[4],clk);
    dff d5(temp_out[5],in[5],clk);
    dff d6(temp_out[6],in[6],clk);
    dff d7(temp_out[7],in[7],clk);

    always @ (posedge clk)
        begin
            out = temp_out;
        end
endmodule
```

Output Register

Code

```
module Register_output(out,in,clk);
    output reg [8:0] out;
    input [8:0] in;
    input clk;
    wire [8:0] temp_out;

    dff d0(temp_out[0],in[0],clk);
    dff d1(temp_out[1],in[1],clk);
    dff d2(temp_out[2],in[2],clk);
    dff d3(temp_out[3],in[3],clk);
    dff d4(temp_out[4],in[4],clk);
    dff d5(temp_out[5],in[5],clk);
    dff d6(temp_out[6],in[6],clk);
    dff d7(temp_out[7],in[7],clk);
    dff d8(temp_out[8],in[8],clk);

    always @ (posedge clk)
        begin
            out = temp_out;
        end
endmodule
```

The Whole System Structurally



Code

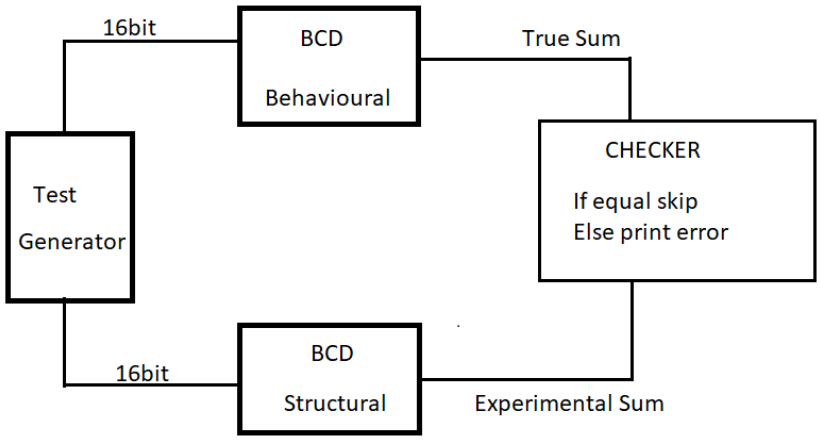
```
module System(A,B,clk,Result);
    input [7:0] A,B;
    reg Carryout;
    output reg [8:0] Result;
    reg [7:0] Aout,Bout;
    reg tempCout,tempCin;
    input clk;
    reg [3:0]SL,SM,AL,BL,AM,BM;
    reg [8:0] tempResult;

    Register_input IA(Aout,A,clk);
    Register_input IB(Bout,B,clk);

    assign AL = Aout[3:0];
    assign BL = Bout[3:0];
    assign AM = Aout[7:4];
    assign BM = Bout[7:4];

    BCD_Adder1bit D1(AL,BL,1'b0,SL,tempCout);

    assign tempCin = tempCout;

    BCD_Adder1bit D2(AM,BM,tempCin,SM,Carryout);

    always @ (posedge clk)
        begin
            assign tempResult = {Carryout,SM,SL} ;
        end

    Register_output O(Result,tempResult,clk);

endmodule
```

BCD Tester for Stage 1

Which is a "BCD" behavioral code

```
module BCD_Tester(A,B,Result);

      input [7:0] A,B;
      output reg [8:0] Result;
      reg [7:0] Sum;
      reg Cout;

      reg [3:0] AL,AM,BL,BM,SL,SM;
      reg temp_carry;
      reg [4:0] temp_SL,temp_SM;

      assign AL = A[3:0];
      assign BL = B[3:0];
      assign AM = A[7:4];
      assign BM = B[7:4];

      always @ (*)

            begin
                  temp_SL = AL + BL + 0;

                  if(temp_SL > 9)

                        begin

                              temp_SL = temp_SL + 6;
                              temp_carry = 1;
                              SL = temp_SL[3:0];

                        end

                  else

                        begin

                              temp_carry = 0;
                              SL = temp_SL[3:0];

                        end

                  temp_SM = AM + BM + temp_carry;

                  if(temp_SM > 9)

                        begin

                              temp_SM = temp_SM + 6;
                              Cout = 1;
                              SM = temp_SM[3:0];

                        end

                  else

                        begin

                              Cout = 0;
                              SM = temp_SM[3:0];

                        end

                        Sum = {SM,SL};
            #700ns    Result = {Cout,Sum};

                  end

endmodule
```

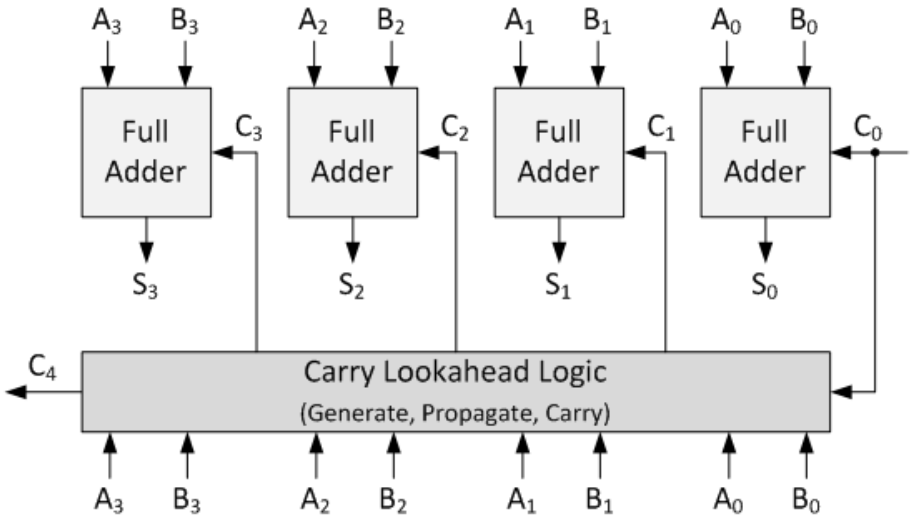
Stage 1



Code

```
module Stage_1();
    reg [7:0] A,B;
    reg clk;
    wire [8:0] True_Result,Exp_Result;

    System S1(A,B,clk,Exp_Result); //Experimental Sum
    BCD_Tester T1(A,B,True_Result);//True Sum

    always  #(100ns) clk = ~clk;

        initial

            begin

                clk = 0;
                {B,A} = 16'b0000000000000000;
                #700ns  {B,A} = 16'b0000000000000000;

                repeat(65535)

                #(900ns) {B,A} = {B,A} + 16'b0000000000000001;

                if(Exp_Result != True_Result)

                    begin

                        $display ("Error! at %b",True_Result);

                    end

        end
endmodule
```

Carry look ahead adder (4-bit)



Code

```
module carry_lookahead_adder_4_bit(i_add1, i_add2, o_result);

  input [3:0] i_add1, i_add2;
  output [4:0] o_result;
  wire [4:0]    w_C;
  wire [3:0]    w_G, w_P, w_SUM;

  FA_1bit F1(.A(i_add1[0]), .B(i_add2[0]), .Cin(w_C[0]), .Sum(w_SUM[0]), .Cout());

  FA_1bit F2(.A(i_add1[1]), .B(i_add2[1]), .Cin(w_C[1]), .Sum(w_SUM[1]), .Cout());

  FA_1bit F3(.A(i_add1[2]), .B(i_add2[2]), .Cin(w_C[2]), .Sum(w_SUM[2]), .Cout());

  FA_1bit F4(.A(i_add1[3]), .B(i_add2[3]), .Cin(w_C[3]), .Sum(w_SUM[3]), .Cout());

  assign w_G[0] = i_add1[0] & i_add2[0];
  assign w_G[1] = i_add1[1] & i_add2[1];
  assign w_G[2] = i_add1[2] & i_add2[2];
  assign w_G[3] = i_add1[3] & i_add2[3];

  assign w_P[0] = i_add1[0] | i_add2[0];
  assign w_P[1] = i_add1[1] | i_add2[1];
  assign w_P[2] = i_add1[2] | i_add2[2];
  assign w_P[3] = i_add1[3] | i_add2[3];

  assign w_C[0] = 1'b0;
  assign w_C[1] = w_G[0] | (w_P[0] & w_C[0]);
  assign w_C[2] = w_G[1] | (w_P[1] & w_C[1]);
  assign w_C[3] = w_G[2] | (w_P[2] & w_C[2]);
  assign w_C[4] = w_G[3] | (w_P[3] & w_C[3]);

  assign o_result = {w_C[4], w_SUM};

endmodule
```

Look Ahead BCD Adder (1-bit)

Code

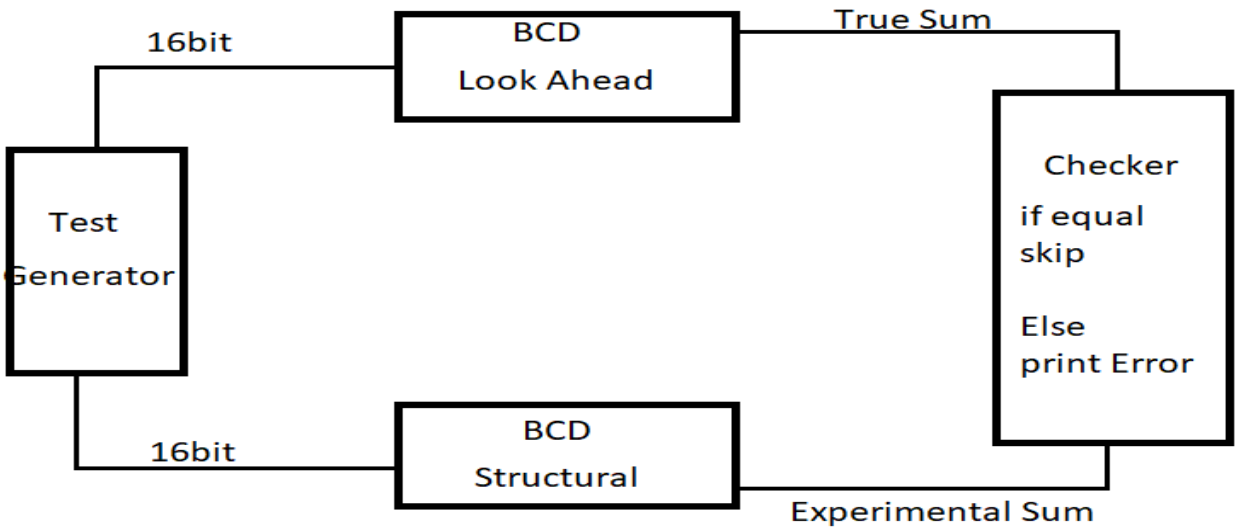
```
module BCD_Adder_LA_1bit(A,B,F);
     input [3:0] A,B;
     output [4:0] F;
     reg [3:0] tempF;
     reg [3:0] Z,S;
     reg tempc;

     carry_lookahead_adder_4_bit BA1(A,B,S);

     always @(*)
     if( S > 4'b1001)
          begin
               assign Z = 4'b0110;
               assign tempc = 1'b1;
          end
     else
          begin
               assign Z = 4'b0000;
               assign tempc = 1'b0;
          end

     carry_lookahead_adder_4_bit BA2(S,Z,tempF);

     assign F={tempc,tempF};
endmodule
```

Look Ahead BCD Adder (2-bit)

Code

```
module BCD_Adder_LA_2bit(A,B,Result);

     input [7:0] A,B;
     output [8:0] Result;
     reg [4:0] SL,SM;
     reg tens,hunds;

     BCD_Adder_LA_1bit B1(A[3:0],B[3:0],SL);

     assign tens = SL[4];

     BCD_Adder_LA_1bit B2({A[7:4]+tens},B[7:4],SM);

     assign hunds = SM[4];

     assign Result = {hunds,SM[3:0],SL[3:0]};


endmodule
```

Stage2



Code

```
module Stage_2();
    reg [7:0] A,B;
    reg clk;
    wire [8:0] True_Result,Exp_Result;

    System S1(A,B,clk,Exp_Result); //Experimental Sum
    BCD_Adder_LA_2bit T1(A,B,True_Result);//True Sum

    always  #(100ns) clk = ~clk;

        initial

            begin

                clk = 0;
                {B,A} = 16'b0000000000000000;
                #700ns  {B,A} = 16'b0000000000000000;

                repeat(65535)

                #(900ns) {B,A} = {B,A} + 16'b0000000000000001;

                if(Exp_Result != True_Result)

                    begin

                        $monitor ("Error! at %b",True_Result);

                    end

        end
endmodule
```

Results

Stage 1

For this stage I've tried a lot of delay values until I reached the one which gives us the results we need

These delay values are

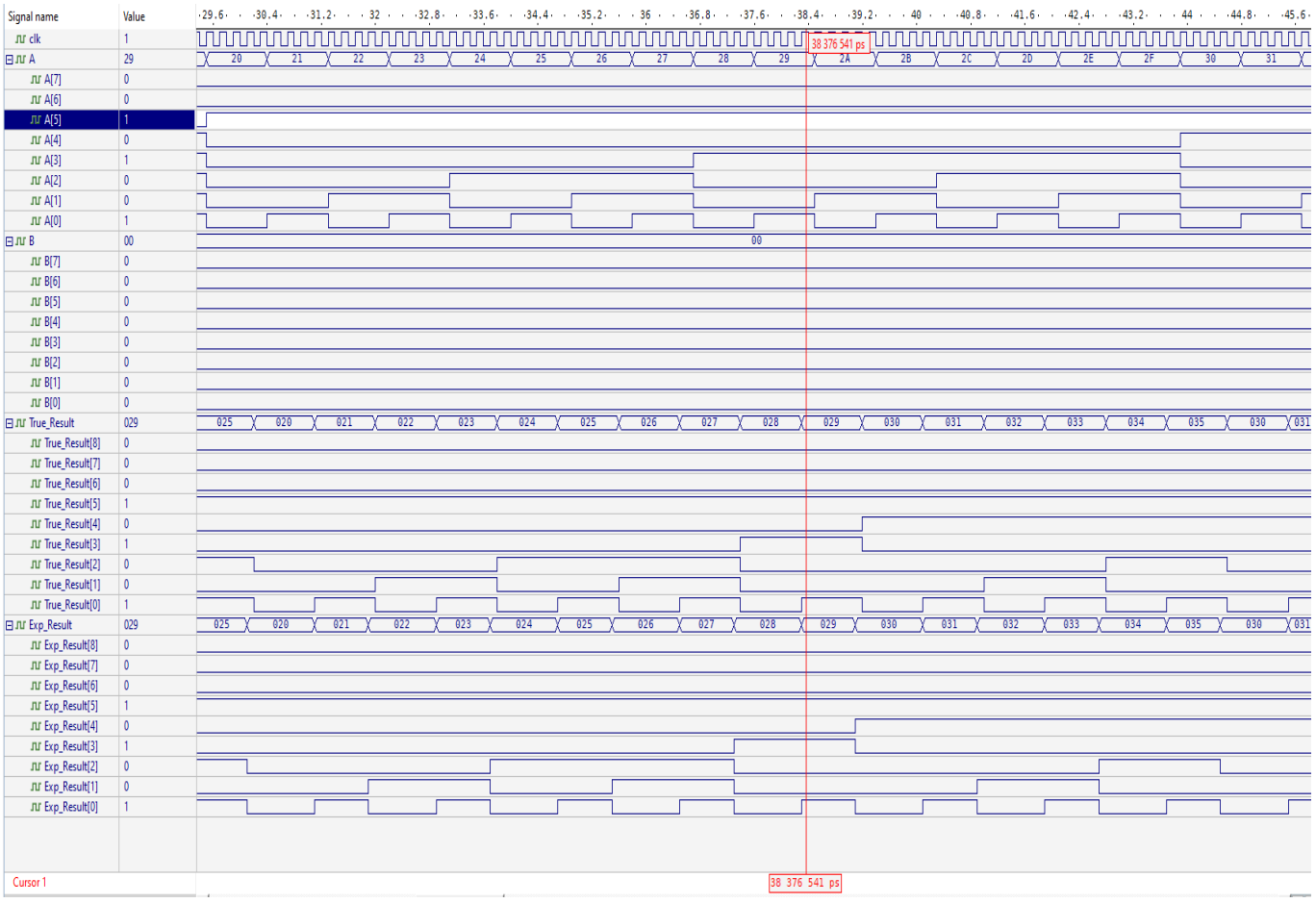
700ns in ~BCD_Tester~ Result

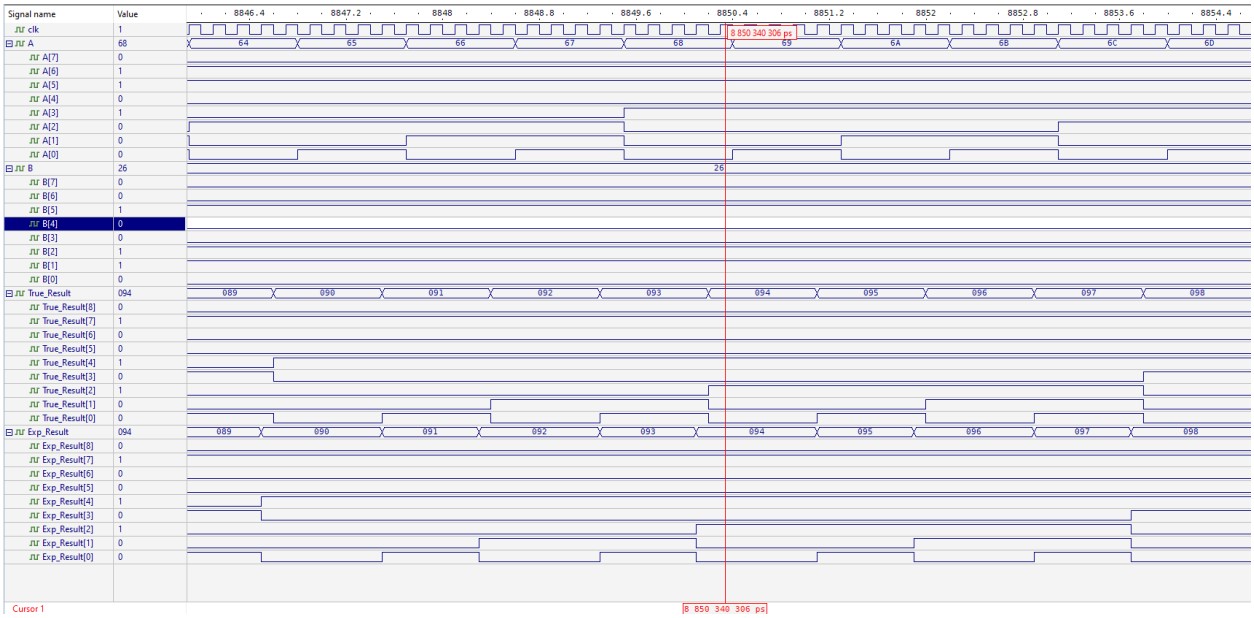
100ns in clk

700ns for {B,A} = 16'b0000000000000000;

900ns for {B,A} = {B,A} + 16'b0000000000000001;

Screenshots





Error

After changing delays from 700 ns, an error occur in checker

```
run 9000000 ns
# KERNEL: Error! at 101011011
# KERNEL: stopped at time: 9 ms

 101101
```

Stage 2

For this stage I've tried a lot of delay values until I reached the one which gives us the results we need

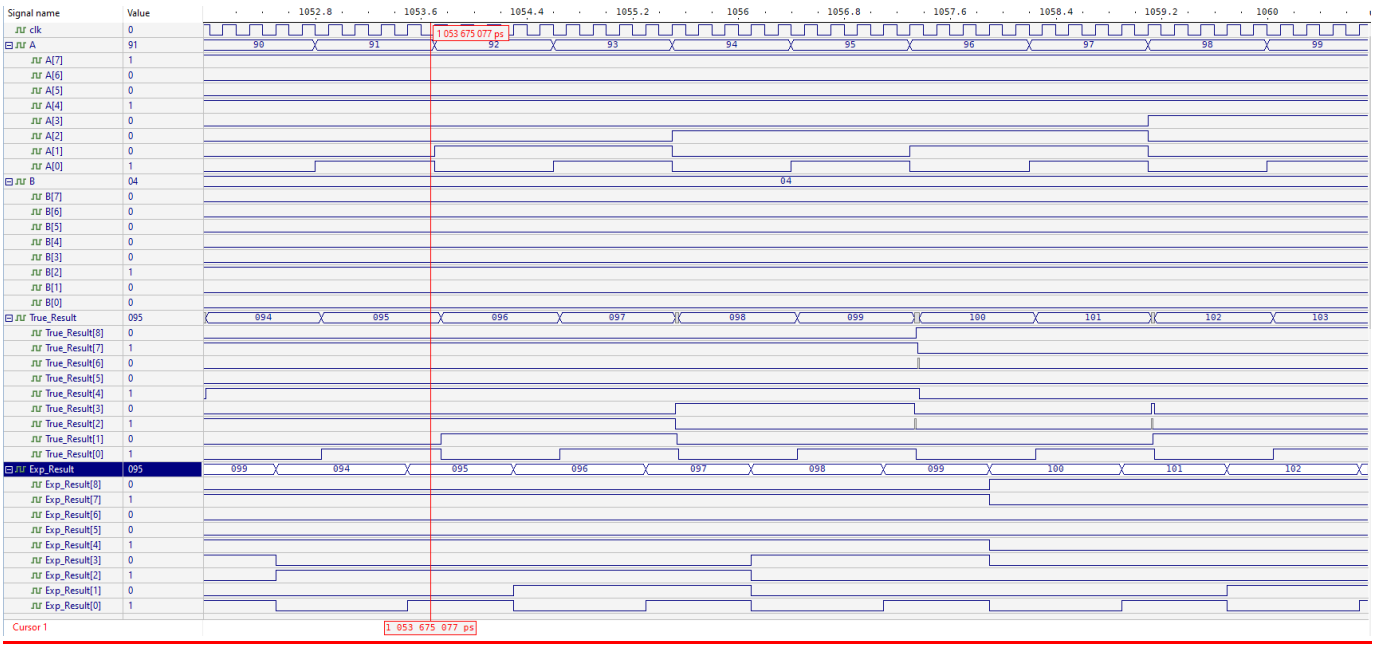
These delay values are

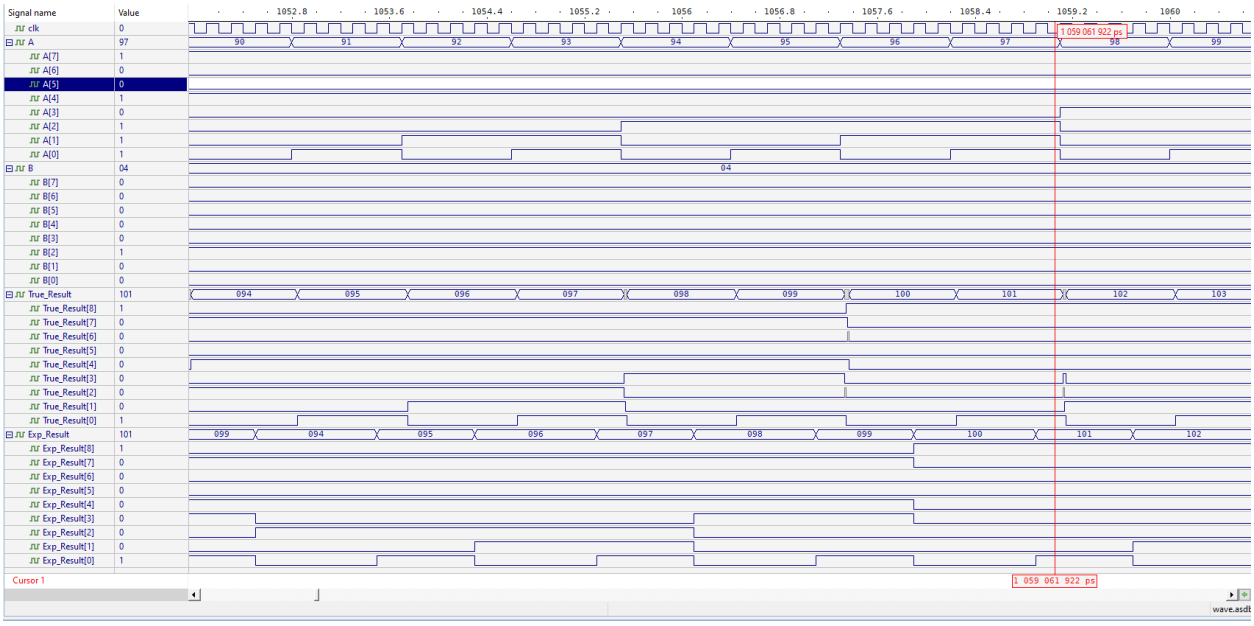
100ns in clk

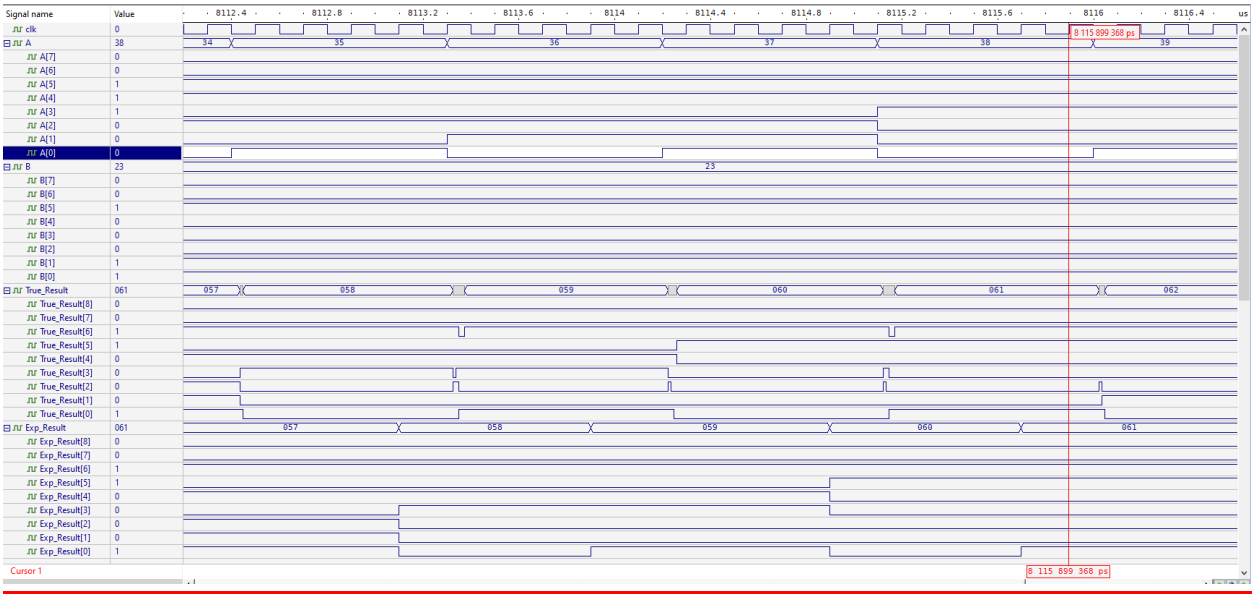
700ns for {B,A} = 16'b0000000000000000;

900ns for {B,A} = {B,A} + 16'b0000000000000001;

Screenshots




Signal name
Value
clk
A
A[7]
A[6]
A[5]
A[4]
A[3]
A[2]
A[1]
A[0]
B
B[7]
B[6]
B[5]
B[4]
B[3]
B[2]
B[1]
B[0]
True_Result
True_Result[8]
True_Result[7]
True_Result[6]
True_Result[5]
True_Result[4]
True_Result[3]
True_Result[2]
True_Result[1]
True_Result[0]
Exp_Result
Exp_Result[8]
Exp_Result[7]
Exp_Result[6]
Exp_Result[5]
Exp_Result[4]
Exp_Result[3]
Exp_Result[2]
Exp_Result[1]
Exp_Result[0]
Cursor 1
1 059 061 922 ps
wave.asdb


Signal name
Value
clk
A
A[7]
A[6]
A[5]
A[4]
A[3]
A[2]
A[1]
A[0]
B
B[7]
B[6]
B[5]
B[4]
B[3]
B[2]
B[1]
B[0]
True_Result
True_Result[8]
True_Result[7]
True_Result[6]
True_Result[5]
True_Result[4]
True_Result[3]
True_Result[2]
True_Result[1]
True_Result[0]
Exp_Result
Exp_Result[8]
Exp_Result[7]
Exp_Result[6]
Exp_Result[5]
Exp_Result[4]
Exp_Result[3]
Exp_Result[2]
Exp_Result[1]
Exp_Result[0]
Cursor 1
8 115 899 368 ps
us

Conclusion

- We can construct any circuit, system, design we need in many ways.
- The small delays on and, or, xor….etc gates have a huge future effects on the results of the system design.
- Look ahead adders have more accurate, less delay results we get.

Appendix

```
module FA_1bit(A,B,Cin,Sum,Cout);

        input A,B,Cin;
        output Sum,Cout;
        wire w1,w2,w3;

        and #(8ns) (w1,A,B);
        xor    #(12ns) (w2,A,B);
        and #(8ns) (w3,w2,Cin);
        xor #(12ns) (Sum,w2,Cin);
        or  #(8ns) (Cout,w1,w3);

endmodule

//////////////////////////////////////////////

module BinaryAdder_4bit(A,B,Cin,Sum,Cout);

        input [3:0] A,B;
        input Cin;
        output [3:0] Sum;
```

```
        output Cout;
        wire C1,C2,C3;

    FA_1bit F1(A[0],B[0],Cin,Sum[0],C1);
    FA_1bit F2(A[1],B[1],C1,Sum[1],C2);
    FA_1bit F3(A[2],B[2],C2,Sum[2],C3);
    FA_1bit F4(A[3],B[3],C3,Sum[3],Cout);


endmodule


////////////////////////////////////////////

module BCD_Adder1bit(A,B,Cin,F,OutputCarry);
    input [3:0] A,B;
    input Cin;
    output [3:0] F;
    output OutputCarry;
        wire [3:0] Z,S;
        wire k,w1,w2,w3;
```

```
    BinaryAdder_4bit BA1(A,B,Cin,Z,k);

        and #(8ns) (w1,Z[2],Z[3]);
    and #(8ns) (w2,Z[1],Z[3]);
    or  #(8ns) (OutputCarry,k,w1,w2);

        assign S = {1'b0,OutputCarry,OutputCarry,1'b0};
        BinaryAdder_4bit BA2(Z,S,0,F,w3);
endmodule

////////////////////////////////////////////////

module dff(Q,D,clk);
        input D,clk;
        output reg Q;

        always @(posedge clk)
                begin
                        Q <= D;
                end
endmodule
```

```
//////////////////////////////////////////////

module Register_input(out,in,clk);
	output reg [7:0] out;
	input [7:0] in;
	input clk;
	wire [7:0] temp_out;

	dff d0(temp_out[0],in[0],clk);
	dff d1(temp_out[1],in[1],clk);
	dff d2(temp_out[2],in[2],clk);
	dff d3(temp_out[3],in[3],clk);
	dff d4(temp_out[4],in[4],clk);
	dff d5(temp_out[5],in[5],clk);
	dff d6(temp_out[6],in[6],clk);
	dff d7(temp_out[7],in[7],clk);

	always @ (posedge clk)
		begin
			out = temp_out;
		end
endmodule
```

```
//////////////////////////////////////////////

module Register_output(out,in,clk);
	output reg [8:0] out;
	input [8:0] in;
	input clk;
	wire [8:0] temp_out;

	dff d0(temp_out[0],in[0],clk);
	dff d1(temp_out[1],in[1],clk);
	dff d2(temp_out[2],in[2],clk);
	dff d3(temp_out[3],in[3],clk);
	dff d4(temp_out[4],in[4],clk);
	dff d5(temp_out[5],in[5],clk);
	dff d6(temp_out[6],in[6],clk);
	dff d7(temp_out[7],in[7],clk);
	dff d8(temp_out[8],in[8],clk);

	always @ (posedge clk)
		begin
			out = temp_out;
```

```
            end
endmodule

////////////////////////////////////////////

module System(A,B,clk,Result);
    input [7:0] A,B;
    reg Carryout;
    output reg [8:0] Result;
    reg [7:0] Aout,Bout;
    reg tempCout,tempCin;
    input clk;
    reg [3:0]SL,SM,AL,BL,AM,BM;
    reg [8:0] tempResult;

    Register_input IA(Aout,A,clk);
    Register_input IB(Bout,B,clk);

    assign AL = Aout[3:0];
    assign BL = Bout[3:0];
    assign AM = Aout[7:4];
```

```
        assign BM = Bout[7:4];


        BCD_Adder1bit D1(AL,BL,1'b0,SL,tempCout);

        assign tempCin = tempCout;

        BCD_Adder1bit D2(AM,BM,tempCin,SM,Carryout);

        always @ (posedge clk)
                begin
                        assign tempResult = {Carryout,SM,SL} ;
                end

        Register_output O(Result,tempResult,clk);

endmodule

////////////////////////////////////////////

module BCD_Tester(A,B,Result);
```

```
input [7:0] A,B;
output reg [8:0] Result;
reg [7:0] Sum;
reg Cout;

reg [3:0] AL,AM,BL,BM,SL,SM;
reg temp_carry;
reg [4:0] temp_SL,temp_SM;

assign AL = A[3:0];
assign BL = B[3:0];
assign AM = A[7:4];
assign BM = B[7:4];

always @ (*)

        begin
                temp_SL = AL + BL + 0;

                if(temp_SL > 9)

                        begin
```

```
            temp_SL = temp_SL + 6;
            temp_carry = 1;
            SL = temp_SL[3:0];

        end

else

        begin

            temp_carry = 0;
            SL = temp_SL[3:0];

        end

temp_SM = AM + BM + temp_carry;

if(temp_SM > 9)

        begin
```

```
                temp_SM = temp_SM + 6;
                Cout = 1;
                SM = temp_SM[3:0];

            end

        else

            begin

                Cout = 0;
                SM = temp_SM[3:0];

            end

            Sum = {SM,SL};
    #700ns  Result = {Cout,Sum};

        end

endmodule
```

```
//////////////////////////////////////////////

module Stage_1();
        reg [7:0] A,B;
        reg clk;
        wire [8:0] True_Result,Exp_Result;

        System S1(A,B,clk,Exp_Result); //Experimental Sum
        BCD_Tester T1(A,B,True_Result);//True Sum

        always  #(100ns) clk = ~clk;

                initial

                        begin

                                clk = 0;
                                {B,A} = 16'b0000000000000000;
                                #700ns      {B,A} = 16'b0000000000000000;

                                repeat(65535)
```

```
                #(900ns) {B,A} = {B,A} + 16'b0000000000000001;

                if(Exp_Result != True_Result)

                    begin

                        $display ("Error! at %b",True_Result);

                    end

            end

endmodule

//////////////////////////////////////////////

module carry_lookahead_adder_4_bit(i_add1, i_add2, o_result);

  input [3:0] i_add1, i_add2;
  output [4:0] o_result;
  wire [4:0]    w_C;
  wire [3:0]    w_G, w_P, w_SUM;
```

```
  FA_1bit F1(.A(i_add1[0]), .B(i_add2[0]), .Cin(w_C[0]),
.Sum(w_SUM[0]), .Cout());

  FA_1bit F2(.A(i_add1[1]), .B(i_add2[1]), .Cin(w_C[1]),
.Sum(w_SUM[1]), .Cout());

  FA_1bit F3(.A(i_add1[2]), .B(i_add2[2]), .Cin(w_C[2]),
.Sum(w_SUM[2]), .Cout());

  FA_1bit F4(.A(i_add1[3]), .B(i_add2[3]), .Cin(w_C[3]),
.Sum(w_SUM[3]), .Cout());

  assign w_G[0] = i_add1[0] & i_add2[0];
  assign w_G[1] = i_add1[1] & i_add2[1];
  assign w_G[2] = i_add1[2] & i_add2[2];
  assign w_G[3] = i_add1[3] & i_add2[3];

  assign w_P[0] = i_add1[0] | i_add2[0];
  assign w_P[1] = i_add1[1] | i_add2[1];
  assign w_P[2] = i_add1[2] | i_add2[2];
  assign w_P[3] = i_add1[3] | i_add2[3];
```

```
  assign w_C[0] = 1'b0;
  assign w_C[1] = w_G[0] | (w_P[0] & w_C[0]);
  assign w_C[2] = w_G[1] | (w_P[1] & w_C[1]);
  assign w_C[3] = w_G[2] | (w_P[2] & w_C[2]);
  assign w_C[4] = w_G[3] | (w_P[3] & w_C[3]);

  assign o_result = {w_C[4], w_SUM};

endmodule

//////////////////////////////////////////////

module BCD_Adder_LA_1bit(A,B,F);
   input [3:0] A,B;
   output [4:0] F;
        reg [3:0] tempF;
        reg [3:0] Z,S;
        reg tempc;

   carry_lookahead_adder_4_bit BA1(A,B,S);

        always @(*)
```

```
        if( S > 4'b1001)
                begin
                        assign Z = 4'b0110;
                        assign tempc = 1'b1;
                end
        else
                begin
                        assign Z = 4'b0000;
                        assign tempc = 1'b0;
                end


        carry_lookahead_adder_4_bit BA2(S,Z,tempF);

        assign F={tempc,tempF};

endmodule

//////////////////////////////////////////////

module BCD_Adder_LA_2bit(A,B,Result);
```

```
        input [7:0] A,B;
        output [8:0] Result;
        reg [4:0] SL,SM;
        reg tens,hunds;

        BCD_Adder_LA_1bit B1(A[3:0],B[3:0],SL);

        assign tens = SL[4];

        BCD_Adder_LA_1bit B2({A[7:4]+tens},B[7:4],SM);

        assign hunds = SM[4];

        assign Result = {hunds,SM[3:0],SL[3:0]};


endmodule

//////////////////////////////////////////////

module Stage_2();
        reg [7:0] A,B;
```

```
reg clk;
wire [8:0] True_Result,Exp_Result;

System S1(A,B,clk,Exp_Result); //Experimental Sum
BCD_Adder_LA_2bit T1(A,B,True_Result);//True Sum

always  #(100ns) clk = ~clk;

        initial

                begin

                        clk = 0;
                        {B,A} = 16'b0000000000000000;
                        #700ns      {B,A} = 16'b0000000000000000;

                        repeat(65535)

                        #(900ns) {B,A} = {B,A} + 16'b0000000000000001;

                        if(Exp_Result != True_Result)
```

```
                begin

                    $monitor ("Error! at %b",True_Result);

                end

        end

endmodule

////////////////////////////////////////////////
```